دورزدن رویکرد تشخیص دستگاه **Root** شده

دور زدن مکانیزم تشخیص اپلیکیشن

[redacted]

شناسه سند:	A-Android-BypassRootDetection
طبقه بندی سند:	**عادی**
شماره نگارش:	R1.0
تاریخ آخرین ویرایش:	۹۹/۰۱/۳۰
تعداد صفحات:	۱۲

Memory Leaks

www.moreti.ir www.memoryleaks.ir

فهرست مطالب

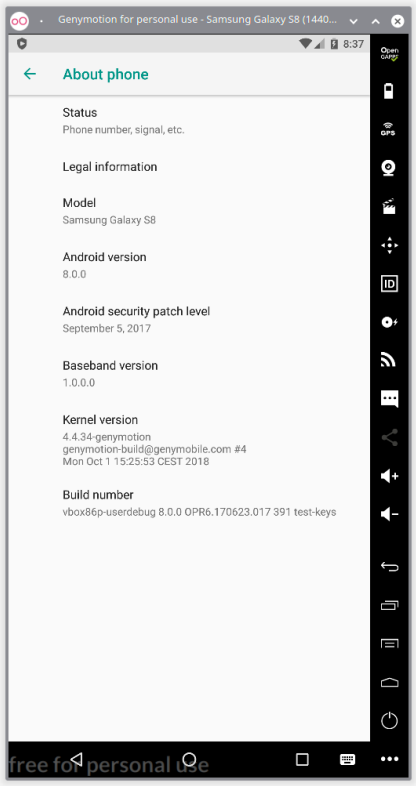
۱. مقدمه

اصولا از آن‌جایی که امنیت اپلیکیشن‌های موبایل از اهمیت بالایی برخوردار است، شرکت‌ها و تیم‌های توسعه از مکانیزم‌های متفاوتی برای تامین امنیت نرم‌افزار تولیدشده، استفاده می‌نمایند. تکنیک‌هایی که مهاجم را از هدف مخرب خود دور می‌سازد. مکانیزم‌های امنیتی که در اپلیکیشن‌ها انجام می‌پذیرند، هرکدام دارای پیچیدگی خاصی می‌باشند. به عنوان مثال یکی از مکانیزم‌های امنیتی که هم در برنامه‌های کاربردی Desktop و هم موبایل توسط توسعه‌دهنده‌ها شایع است، استفاده از مکانیزم‌های مبهم‌سازی[1] می‌باشد. همان‌طور که ممکن است نام این مکانیزم امنیتی را شنیده باشید، با استفاده از این رویکرد، کد منبع برنامه با هدف تغییر تمامی رشته‌های مورد استفاده در کد برنامه که توسط مهاجم در فاز تست استاتیک آنالیز می‌شود، مبهم می‌گردد. استفاده از این تکنیک در بین اپلیکیشن‌های تحت پلتفرم اندروید و iOS به شدت توصیه‌شده است. چرا که مهاجم با ابزارهای رایج می‌تواند به راحتی به کد منبع دسترسی پیدا کند. هدف ما در این مقاله دورزدن یکی از مکانیزم‌های معمول در اپلیکیشن‌های اندروید به نام Root Detection در برنامه [redacted] است که فایل apk آن توسط [redacted] قرار گرفته است.

در بسیاری از موارد، این که اپلیکیشن مورد نظر چه کاربردی برای کاربران دارد و برای چه هدفی پیاده-سازی شده است از اهمیت بالایی برخوردار است. چرا که با استفاده از تست‌های Logic می‌توان آسیب-پذیری‌هایی را یافت که از کارکرد اپلیکیشن سوء استفاده گردد و حتی منجر به گرفتن دسترسی به سرورهای Backend اپلیکیشن شود و به طور کامل کنترل سرورها را به دست گرفت. اما همان‌طور که در موضوع این گزارش نیز مطرح گردیده، هدف ما از تست این برنامه فقط دورزدن مکانیزم Root Detection می‌باشد. به این دلیل که تست ما بر روی موبایل اپلیکیشن از نوع استاتیک می‌باشد، فلذا داشتن اطلاعات اضافه درباره کارکرد اپلیکیشن کمکی به ما نمی‌کند.

[1] Obfuscation

در پروسه تست استاتیک اپلیکیشن اندرویدی مذکور، ابزارهای زیر مورد نیاز است که برای هرکدام توضیح مختصری ارائه گردیده است. برای انجام فرآیند تست نفوذ، هرچند که امکان تست و آنالیز بر روی سیستم عامل ویندوز نیز امکان‌پذیر است اما استفاده از سیستم عامل لینوکس توصیه می‌گردد. از بین توزیع‌های سیستم عامل محبوب لینوکس می‌توان از Kali Linux استفاده نمود، اما سیستم عامل Santoku Linux مخصوص تست نفوذ برای اپلیکیشن‌های اندرویدی تعبیه گردیده است که ابزارهای مورد استفاده در این مقاله به صورت پیش‌فرض در این سیستم‌عامل نصب می‌باشند.



Genymotion .۱.۱

نرم‌افزار Genymotion یک ابزار با هدف شبیه‌سازی دستگاه اندرویدی می‌باشد. ابزارهای شبیه‌ساز دیگری نیز مانند NOX وجود دارند، ولی این ابزار به دلیل نصب آسان مورد استفاده قرار گرفته است. این برنامه از نرم‌افزار پرکاربرد VirtualBox استفاده می‌نماید. دستگاهی که ما در محیط Genymotion نصب کردیم، Samsung Galaxy S8 با مشخصات روبرو می‌باشد.

همانطور که مشاهده می‌نمایید، می‌توانید نسخه‌های مختلف اندرویدی را نصب نمایید که هرکدام دارای API مشخص می‌باشند.

Apktool .۲.۱

یکی از مهم‌ترین ابزارها در حوزه تست نفوذ اپلیکیشن‌های اندرویدی می‌باشد که در لینوکس و ویندوز قابلیت نصب دارد. با استفاده از این ابزار می‌توان اپلیکیشن اندرویدی را اصطلاحا Reengineer نمود. با استفاده از این ابزار می‌توان هر فایل apk را Decompile نمود و سپس بعد از تغییرات انجام شده در کد، آن اپلیکیشن را Build نمود. این ابزار یکی از کاربردی‌ترین ابزارهای تولیدشده در حوزه تست نفوذ اندروید

می‌باشد که تمامی متخصصان تست نفوذ اندروید از آن استفاده می‌نمایند. در این پروژه ما از نسخه 2.4.1 این برنامه استفاده می‌نماییم. صفحه گیت‌هاب این ابزار را از این طریق می‌توانید مشاهده نمایید.

۱.۳. Adb

Adb یکی از ابزارهای گروه android-tools می‌باشد. اگر شما اقدام به نصب پکیج android-tools نمایید، این ابزار برای شما نصب خواهدشد. هرچند که می‌توانید به صورت جداگانه نیز آن را نصب نمایید. با استفاده از این ابزار شما می‌توانید به دستگاه اندرویدی که در محیط شبیه‌ساز Genymotion اجرا نموده‌اید و یا دستگاه فیزیکی‌تان، دسترسی Shell داشته باشید و یا اپلیکیشن‌ها را نصب یا حذف کنید و یا بین دستگاه اندرویدی و سیستم عامل Host تان فایل‌ها را Share نمایید. قابلیت‌های مختلف دیگری را این ابزار به کاربران می‌دهد که می‌توانید درباره آن‌ها در این صفحه مطالعه نمایید.

۲. دورزدن مکانیزم امنیتی مربوطه

از آن‌جایی که فایل‌های با فرمت apk، فایل‌هایی به صورت فشرده‌شده می‌باشند، می‌توان به راحتی با استفاده از دستور زیر فایل را از حالت فشرده خارج نمود تا جزییات بیشتری مشاهده گردد.

```
$ unzip original_file.apk -d ./original_file
```

```
total 14M
4.0K drwxr-xr-x 13 moreti moreti 4.0K Mar 22 18:33 ./
4.0K drwxr-xr-x  4 moreti moreti 4.0K Mar 22 19:11 ../
8.0K -rw-rw-rw-  1 moreti moreti 6.8K Nov 30  1979 AndroidManifest.xml
4.0K drwxr-xr-x  5 moreti moreti 4.0K Mar 22 18:33 assets/
4.2M -rw-r--r--  1 moreti moreti 4.2M Nov 30  1979 classes2.dex
8.4M -rw-r--r--  1 moreti moreti 8.4M Nov 30  1979 classes.dex
4.0K drwxr-xr-x  3 moreti moreti 4.0K Mar 22 18:33 com/
4.0K drwxr-xr-x  2 moreti moreti 4.0K Jan  1  1970 error_prone/
4.0K drwxr-xr-x  3 moreti moreti 4.0K Mar 22 18:33 javax/
4.0K drwxr-xr-x  2 moreti moreti 4.0K Jan  1  1970 jsr305_annotations/
4.0K drwxr-xr-x  3 moreti moreti 4.0K Mar 22 18:33 junit/
4.0K drwxr-xr-x  4 moreti moreti 4.0K Mar 22 18:33 lib/
 12K -rw-r--r--  1 moreti moreti  12K Nov 30  1979 LICENSE-junit.txt
4.0K drwxr-xr-x  2 moreti moreti 4.0K Jan  1  1970 META-INF/
4.0K drwxr-xr-x  3 moreti moreti 4.0K Mar 22 18:33 org/
 36K -rw-r--r--  1 moreti moreti  34K Nov 30  1979 publicsuffixes.gz
4.0K drwxr-xr-x 36 moreti moreti 4.0K Mar 22 18:33 res/
404K -rw-rw-rw-  1 moreti moreti 403K Nov 30  1979 resources.arsc
4.0K drwxr-xr-x  3 moreti moreti 4.0K Mar 22 18:33 third_party/
```

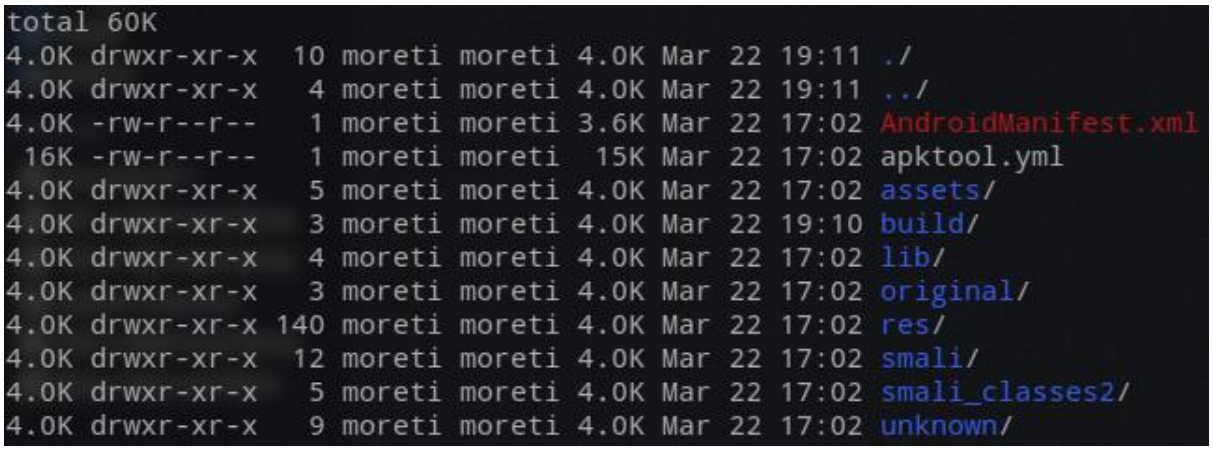
تصویر ۱ – جزییات پوشه original_file

توضیحات درخصوص جزییات پوشه‌ها و فایل‌های موجود در apk های موجود در حوزه‌ی موضوع این مقاله نیست. فقط به این مورد اشاره می‌نماییم که فایل های AndroidManifest.xml و classes.dex و classes2.dex برای ما از نظر امنیتی اهمیتی بالاتری نسبت به بقیه پوشه‌ها و فایل‌ها دارند. در خصوص پوشه META-INF نیز ذکر این مورد خالی از لطف نیست که فایل‌های مربوط به امضای معتبری که بر apk انجام شده است، در این پوشه قرار دارد. برای مابقی مواردی که در این دایرکتوری قرار دارد می‌توان با استفاده از جستجو و مطالعه بیشتر اطلاعات کامل‌تری کسب کرد.

با اجرای دستور زیر فایل apk را decompile می‌نماییم :

$ apktool d original_file.apk –o apktool_out/

با بازنمودن پوشه apktool_out/ می‌توان فایل‌های درون برنامه را مشاهده نمود.



تصویر ۲ – جزییات پوشه apktool_out

در خصوص پوشه‌های مشاهده‌شده، برای هدف ما که در اینجا دورزدن مکانیزم تشخیص دستگاه روت‌شده می‌باشد، فقط پوشه‌های smali و smali_classes2 اهمیت دارد. فایل AndroidManifest.xml نشان-دهنده دسترسی‌های مورد نیاز برنامه برای اجرا می‌باشد. که برای این دسترسی‌ها در زمان نصب از کاربر کسب اجازه می‌شود. در این فایل قسمت‌های مختلفی مانند Backup و یا Debug از نظر امنیتی دارای اهمیت می‌باشند.

```
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.camera.flash" android:required="false"/>
<uses-feature android:name="android.hardware.screen.landscape"/>
<application android:allowBackup="false" android:icon="@drawable/logo" android:label="@string/appName" android:name="
android.support.multidex.MultiDexApplication" android:theme="@android:style/Theme.NoTitleBar">
    <meta-data android:name="com.google.android.maps.v2.API_KEY" android:value="@string/google_maps_api_key"/>
    <activity android:label="@string/app_name" android:name="com.isc.view.Main_Activity" android:screenOrientation="portrait"
     android:theme="@android:style/Theme.Light.NoTitleBar"/>
    <activity android:launchMode="singleTask" android:name="com.isc.view.PauseActivity" android:screenOrientation="portrait"
    android:theme="@android:style/Theme.Light.NoTitleBar">
```

تصویر ۳ – Attribute مربوط به Backup

همان‌طور که در تصویر ۳ قابل مشاهده است، در تگ application مقدار android:allowBackup برابر با false می‌باشد. این قابلیت برای انجام عملیات Back-Up گیری و یا Restore کردن اپلیکیشن اندرویدی تعبیه گردیده است. مقدار پیش‌فرض آن، یعنی در حالتی که در فایل AndroidManifest به آن اشاره نشده باشد، برابر true است. این ویژگی قابلیت تهیه نسخه پشتیبان، حتی پشتیبان گیری در سطح full-system که در آن ابزار adb اطلاعات همه سیستم را ذخیره می‌کند، به کاربر می‌دهد. در خصوص هدف ما که دور زدن مکانیزم تشخیص دستگاه روت شده می‌باشد، این قابلیت اهمیتی ندارد. در خصوص

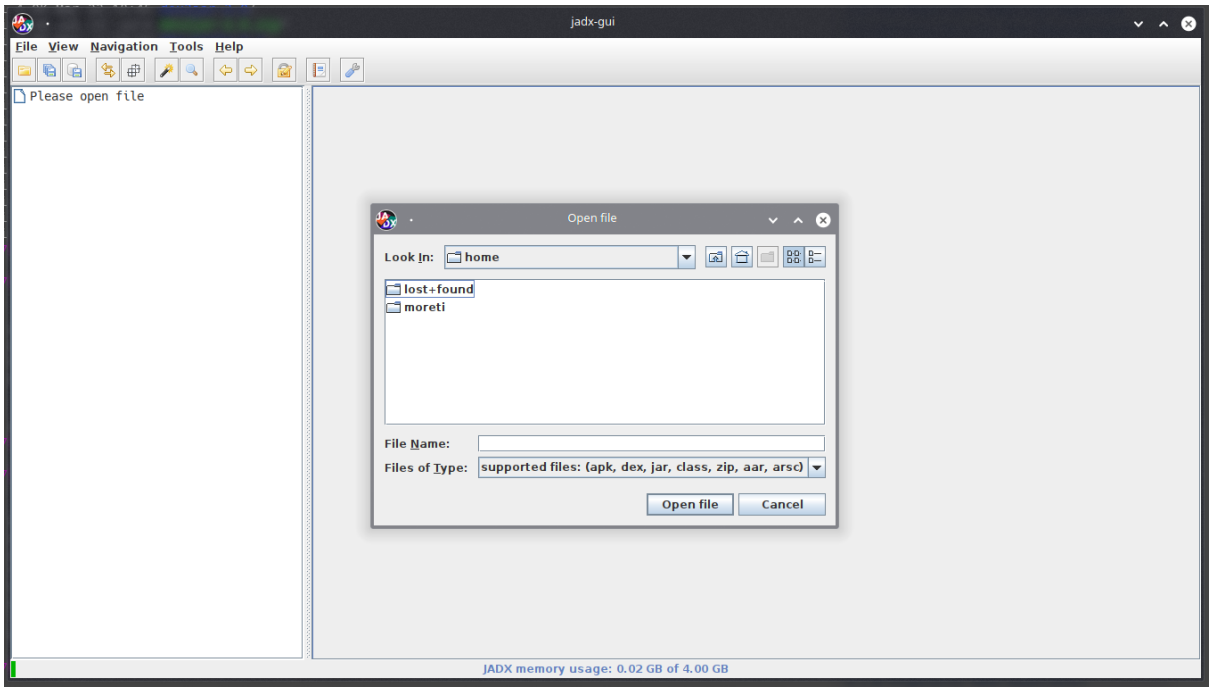
فیلد Debug نیز اگر این قابلیت با مقدار true تنظیم شده باشد، امکان Debug کردن اپلیکیشن توسط ابزارهای خودکار و یا اسکریپت‌های دستی فراهم می‌شود.

اصولاً برای یافتن تابعی در برنامه که وظیفه تشخیص دستگاه روت‌شده را دارد، از روش‌ها و رویکردهای مختلفی می‌توان استفاده نمود. در اینجا ما فقط یکی از روش‌ها را مورد تحلیل و بررسی قرار می‌دهیم ولی در صورتی که فرصت شود در مقاله‌ها و بررسی‌های آینده بقیه تکنیک‌ها نیز درج خواهدگردید. رویکردی که در اینجا مورد نظر ما است، تحلیل استاتیک کد برنامه و یافتن تابعی است که وظیفه تشخیص دستگاه روت شده را دارد. این کار با استفاده از مطالعه کدهای سورس برنامه انجام می‌شود.

همان‌طور که می‌دانید اکثر اپلیکیشن‌های اندرویدی به زبان جاوا نوشته می‌شوند، البته در بعضی از اپلیکیشن‌های به‌روزتر، از کدهای Native هم در آن‌ها استفاده شده است که بعضی مکانیزم‌های امنیتی را در آن پوشش می‌دهند. در اپلیکیشن‌های اندرویدی که به زبان جاوا تولید شده‌اند، بعد از مرحله Decompile معمولا پوشه هایی با پیشوند smali تولید می‌شود. اگر در پوشه‌های مربوطه جستجو کنیم، تعدادی فایل مشاهده می‌گردد که به زبان smali نوشته شده‌اند. زبان smali یک زبان سطح پایین مانند Assembly می‌باشد که برنامه بعد از Decompile شدن به پایین‌ترین سطح ممکن قابل مشاهده است. مثل برنامه‌های تحت Desktop که بعد از Decompile شدن، کدهای برنامه به زبان assembly قابل مشاهده است، در اپلیکیشن‌های اندرویدی نیز بعد از این فرآیند کدهای smali قابل مشاهده هستند. متاسفانه هیچ مستند کامل و دوره آموزشی معتبری درباره زبان smali وجود ندارد و متخصصان حوزه مهندسی معکوس در اپلیکیشن‌های اندرویدی، عمدتا با آزمون و خطا و مشاهده و تحلیل کدهای smali به درک درستی از این زبان دست پیدا می‌نمایند. در صورت درخواست برای آموزش و بیان اصول و کلیات این زبان در صورت نیاز در آینده، مقاله‌ای شکل خواهد گرفت.

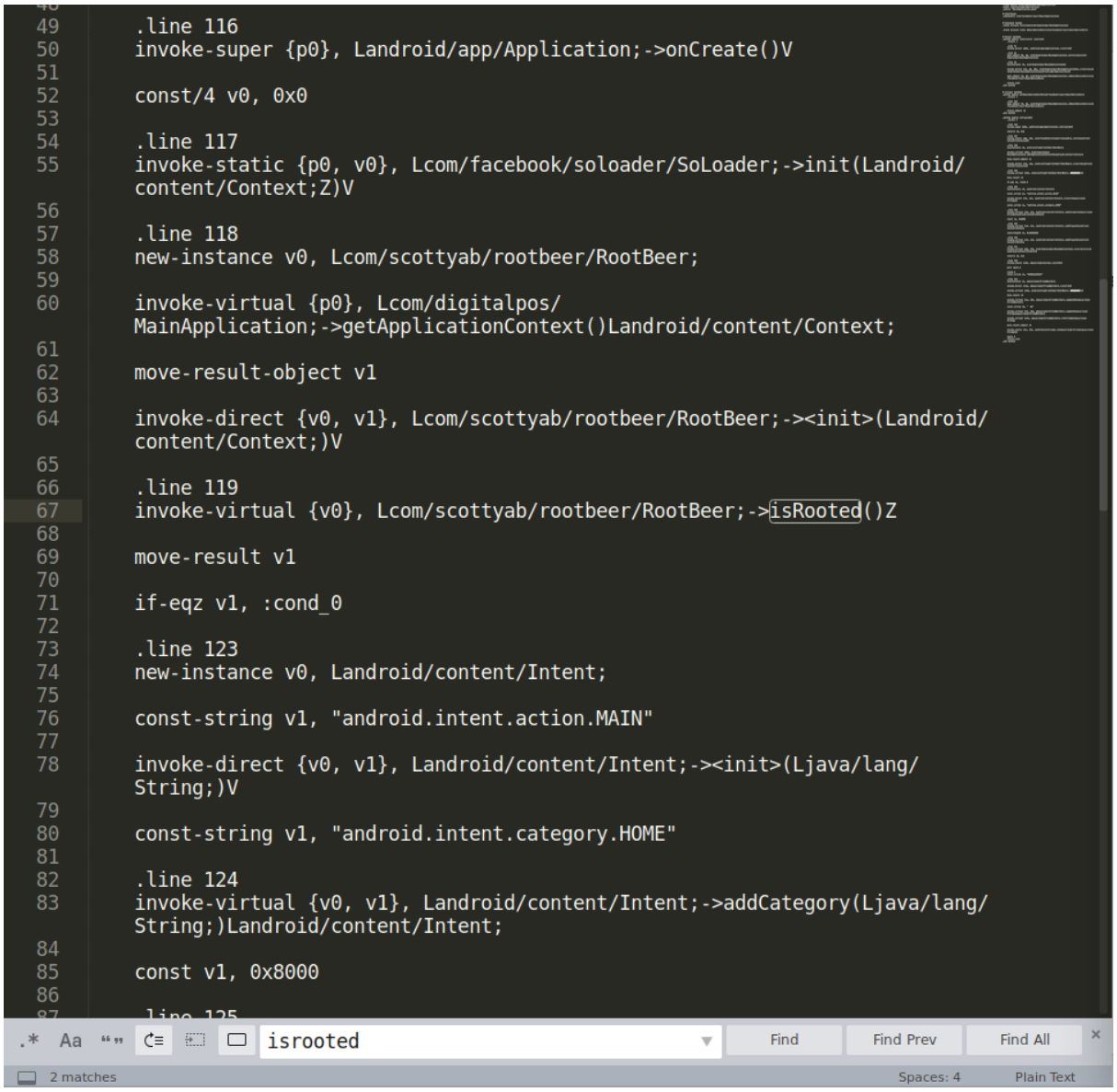
اما سوالی که در این جا مطرح است، این است که آیا می‌توان کار ساده‌تری انجام داد؟ آیا می‌توان از یادگیری زبان جدیدی سر باز زد و انجام فرآیند دورزدن را به روش دیگری انجام داد ؟ جواب مثبت است. ابزاری به اسم jadx تولید شده است که می‌تواند فایلهای apk، dex، jar، class، zip، aar و arsc را باز

نماید و کدهای موجود در این پسوندهای فشرده را افشا نماید. شما می‌توانید برای سادگی کار فایل apk مربوطه را به این اپلیکیشن که محیط GUI نیز دارد، داده و کدها را به راحتی به زبان جاوا مشاهده کنید و با کارایی اپلیکیشن بیشتر آشنا شوید، چرا که نقطه شروع برای همه فرآیندهای تست نفوذ، جمع‌آوری اطلاعات می‌باشد. البته در این برنامه امکان تغییر و Build دوباره وجود ندارد و برای این فرآیند از رویکرد دیگری استفاده می‌شود. ما با استفاده از برنامه jadx-gui می‌توانیم دایرکتوری که مربوط به فایل است را پیداکرده و با یافتن آن فایل بخصوص در پوشه Decompile شده برنامه، فایل انتخاب شده را پیدا نماییم و با تغییر در آن، فرآیند Build را در ادامه انجام دهیم.



تصویر ۴ – محیط برنامه JadX

بعد از پیدا کردن فایل مربوطه به هر یک از روش‌های گفته‌شده در بالا، در نهایت به یک فایل smali می‌رسیم. در تصویر ۵ نمونه فایلی که مربوط به مکانیزم تشخیص دستگاه روت شده است، مشاهده می‌گردد.



تصویر ۵ – فایل MainApplication به زبان smali

تصویر ۵ نشان دهنده تابعی است که در فایل MainApplication وظیفه کنترل دستگاه روت‌شده را دارد. همانطور که قابل مشاهده است، اپلیکیشن مورد نظر از کتابخانه معروف RootBeer استفاده می‌نماید که آدرس فراخوانی آن در خط ۶۷ تصویر بالا قابل مشاهده است و در آن فایل جزییات تابع مورد نظر قابل تحلیل و آنالیز می‌باشد. حال که این فراخوانی را در فایل MainApplication یافتیم، باید نتیجه این فراخوانی را تغییر دهیم. ساده‌ترین راهی که می‌توان انجام داد، تغییر شرط if-eqz به if-nez است. با این تغییر ما شرط برنامه را معکوس کردیم به این صورت که برنامه فقط در محیط روت‌شده قابل بارگذاری و

اجرا باشد. درصورت نیاز می‌توان در فایلی که از آن فراخوانی انجام شده است، تغییرات دیگری انجام داد که برنامه در هر محیطی اجرا شود. بعد از انجام تغییر در شروط برنامه، می‌توان فایل را ذخیره نماییم. به پوشه مادر (پوشه ای که فایل apk در آن قرار دارد) می‌رویم و با استفاده از دستور زیر برنامه را دوباره Build می‌نماییم.

```
$ apktool b ./original_file/ -o edited_file.apk
```

بعد از Build کردن برنامه، برای نصب این اپلیکیشن بر روی دستگاه اندرویدی، بایستی امضای معتبری بر روی این فایل apk قرار گیرد. فرآیند این امضا با اجرای دستورات زیر قابل انجام است.

```
$ keytool –genkey –v –keystore my_release_key.keystore –alias alias_name –keyalg RSA –
    keysize 2048 –validity 3650
$ jarsigner –verbose –sigalg MD5withRSA –digestalg SHA1 –keystore
    my_release_key.keystore edited_file.apk alias_name
```

و در نهایت برای مشاهده امضای اپلیکیشن می‌توان دستور زیر را اجرا نمود.

```
$ jarsigner –verify –verbose edited_file.apk
```

با انتقال فایل edited_file.apk به محیط اندرویدی و نصب آن، فرآیند با موفقیت به پایان می‌رسد و در نهایت ما موفق به نصب اپلیکیشن در محیط روت‌شده می‌باشیم.

۳. چگونه از ما حمایت کنید ؟

با اشتراک گذاشتن این فایل می‌توانید ما را در امر انتشار گزارشات جدید آسیب پذیری‌ها و آموزش‌های امنیتی حمایت نمایید.

باتشکر