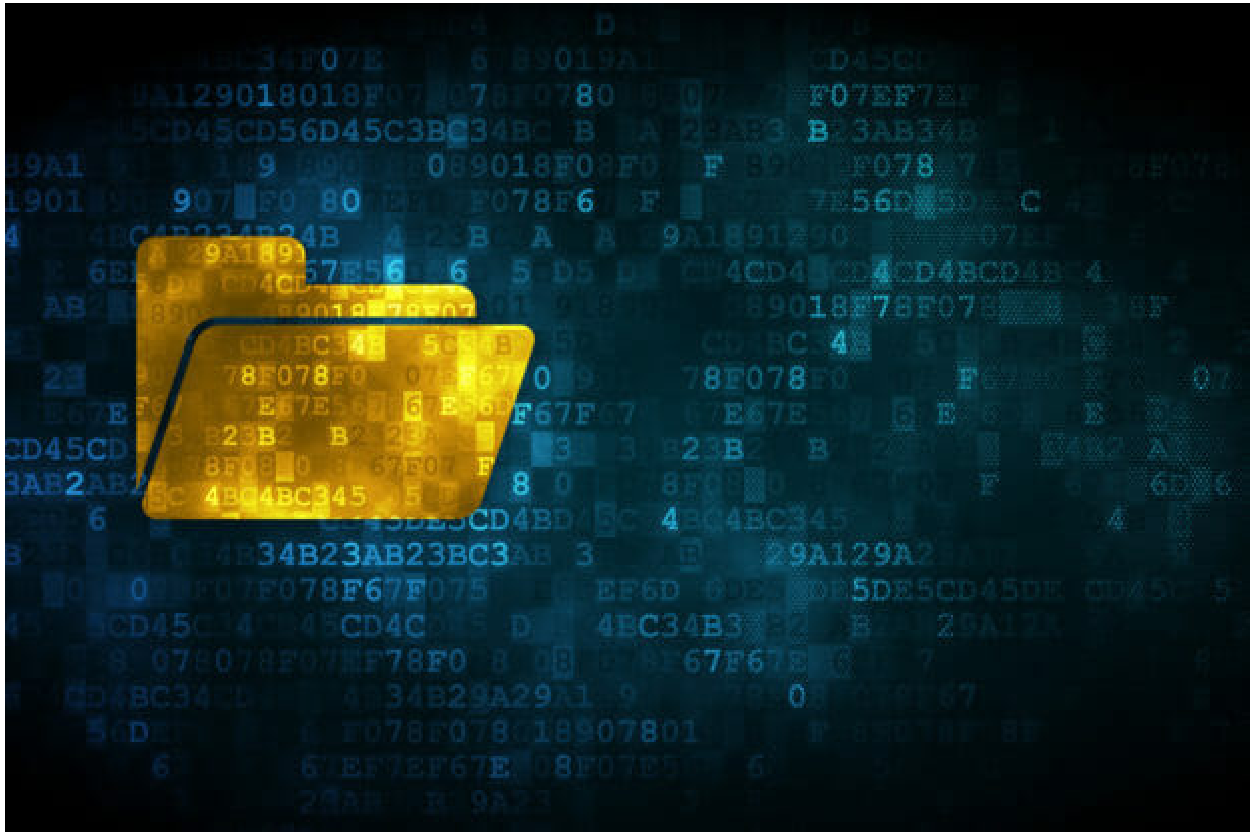
Secplicity®
Security Simplified

Powered by WatchGuard Technologies

Android APK Reverse Engineering: What’s in an APK

September 11, 2019 By Emil Hozan

Privacy - Terms



Before building off the previous post, I wanted to take a moment and clarify the objective of this series. The purpose of this series is to break down the APK reverse engineering process into smaller chunks in order to really appreciate each step. As we progress through the series, I want readers to be able to build off of the previous post and see how all the pieces tie together. Hopefully this yields a greater understanding of the entire process.

Note that none of what I cover is aimed at piracy nor would I ever promote foul behaviors. What I write about is purely educational and I encourage educational growth in different aspects of life. My stance is

Privacy - Terms

from a security research standpoint, learning to better understand how APKs are laid out and what they're all about, but also how malicious apps can exploit their user base.

Moving on, this post will cover what an APK is, how to get ahold of an APK for analysis, and how to go about accessing and interpreting the content within. There are many ways to access the APK file structure, and in many ways is how I did it! Why? Because I wanted to see what each method had to offer and how they stacked up against each other. I also used other online resources to better understand what to expect within this file structure. Lastly, I am analyzing multiple APKs; one is an APK that I developed, so I have it available as a control and can compare it with other samples. These other samples come from Google Play Store.

As a final note for the introduction, I do just about all testing in an isolated virtual network using virtual machines (VMs). There are many reasons I do this but the most important reason is to safely do what I need to without fear of sabotaging my actual machine. That said, it also allows for experience in creating and maintaining a VM infrastructure and I highly suggest seeking out open source options. In my case, I am using Proxmox as the hypervisor and Linux Ubuntu and Windows clients as the test VMs. Some tools work on either, others only on Windows. The tool reviews will be discussed in a future post.

Understanding an APK

Let's start with the obvious question: What is an APK?

Turning to the great oracle, Google.com, let's see what results we get when searching for it. As expected, many links and resources were returned. I will stick with Wikipedia due to its community-editable nature. In reading it, sure enough we get a fair idea of what an APK is.

An *APK* is a package file format used by devices running the Android operating system. It contains all of a program's required code in order to run appropriately; of interest are the manifest file (AndroidManifest.xml; describes the name, version, and other bits of information about the application at hand), .dex files (Java source code compiled to bytecode, translated for Android's Dalvik process VM and-now Android RunTime), certificates (used for security and code signing), and others that we'll poke into with time.

I'm a sucker for checking multiple sources of information and not relying on just one source. To that I say that this Quora question has some great answers adding nice details. I also came across this article that I thought was written quite well. It covers this section's content in more detail (such as the APK build process and what happens behind the scenes) as well as talking about a few other sections in this post.

Acquiring an APK for Analysis

Personally, I like to follow along when reading articles so this is where I will discuss how you can obtain an APK yourself for analysis. There are a few ways to do this, but I will only cover two; a third option is through a physical device that I will not cover in this post. The two options I will cover allow all of this to be done via web, which is more preferred in my virtual environment.

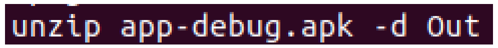
Within my VM, Ubuntu in this case, I have Firefox installed and a corresponding plugin called APK Downloader. You can install it by opening Firefox, clicking on the menu option (the three stacked lines in the top-right corner), and selecting “Add-ons”; alternatively, you can use Ctrl+Shift+A. Within this window is a search bar, type in “apk downloader,” click its name, then “+ Add to Firefox” to complete the installation process. It’s the first result for my query, there are many others that I haven’t tested myself. Another option I came across during my writing for this blog is attributed by this Quora question, and that revealed this resource.

I will say that I like that the latter maintains a readable filename of the downloaded APK, whereas Firefox’s APK Downloader add-on names the APK what appears to be a hash of sorts – definitely not as readable. On the flip side, it didn’t work for me in all cases, as I tried downloading an APK in Ubuntu and that worked fine but not another test client. Try either of them or find other ways yourself to obtain an APK for analysis purposes. Of course, you can also follow Android developer’s introduction guide and create a simple “Hello World” APK for use!

Tapping into the APK File Structure

Hopefully you now have an APK to test with at this point, as this section will cover how to get into the APK file structure and see the content contained within. For this example, I will be using an Ubuntu machine and am going to simply “unzip” the APK with a command line utility conveniently named, you guessed it, unzip!

The command itself is: unzip app-debug.apk -d Out



```
unzip app-debug.apk -d Out
```

Figure 1: unzip Command Line Usage

unzip is the utility; *app-debug.apk* is my Hello World APK; and the *-d* flag specifies which directory to unzip the content into, in this case a directory named *Out*. After doing so, we go from a single archive to multiple files as contrasted between Figures 2 and 3.

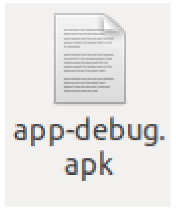


Figure 2: A Simple APK Archive

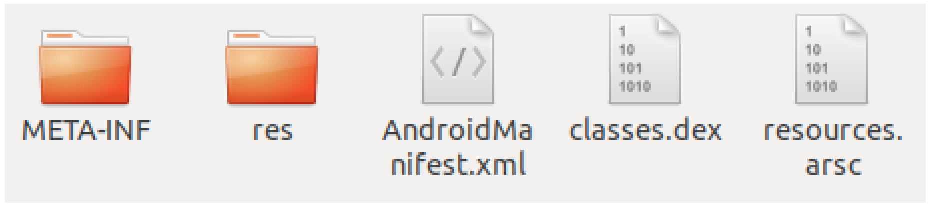


Figure 3: An Unzipped APK Folder Structure

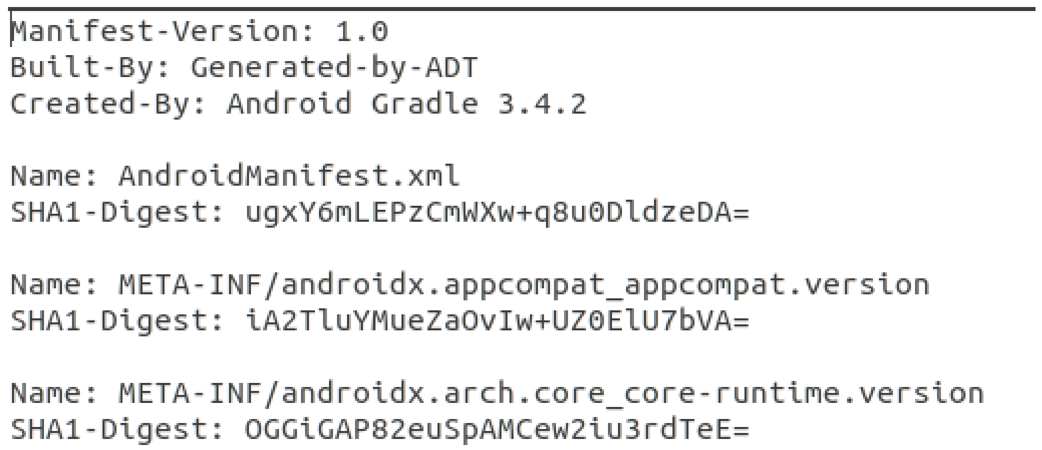
For now, the only content you can open is within “META-INF” and “res,” the others are in binary format and require special processing to open. That didn’t stop me from at least trying to see what it’s all about!

The terminal output is less than useful but a GUI program, Ubuntu’s Text Editor, offers a bit more insight but still not fully legible.

Tying in what we learned in the first section, let’s expand just a bit on the content:

- AndroidManifest.xml
  This is where an app’s permissions and functionalities are documented and identified. We’ll go more into this in a future post, this is where we can spot applications requesting suspicious permissions.
- classes.dex
  Essentially this is the code that makes the APK what it is – it’s the app’s logic. This file needs to be converted from Dalvik instructions into Android bytecode, then decompiled back into Java source code for easier reading.
- resources.arsc
  Wikipedia’s description says that it’s “a file containing precompiled resources, such as binary XML for example.” Having looked at the contents using another tool, this is mainly used behind the scenes such as stylizing appearances or localizing strings into different languages.
- META-INF
  Here is the security, so to speak, in the sense of having hashes available for the written code. The way I understand it is in three parts, each corresponding to the main three files in META-INF:

  - MANIFEST.MF
    This offers a hash for each component of the app; the AndroidManifest.xml, the appcompat_appcompat.version hash, and many other “Name”d files as seen in the Figure. Basically, if there is a change to any named component, rehashing that component will generate a completely different hash – that’s just how hashes work!

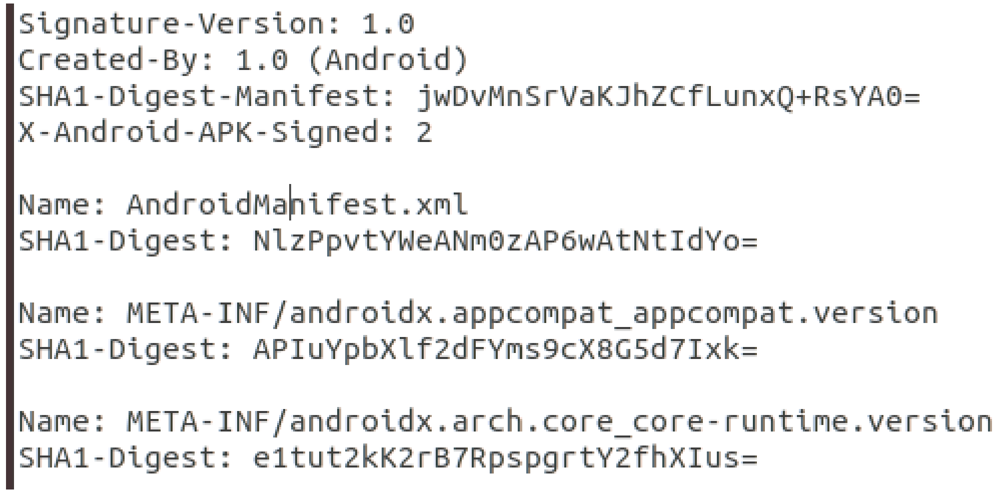


```
Manifest-Version: 1.0
Built-By: Generated-by-ADT
Created-By: Android Gradle 3.4.2

Name: AndroidManifest.xml
SHA1-Digest: ugxY6mLEPzCmWXw+q8u0DldzeDA=

Name: META-INF/androidx.appcompat_appcompat.version
SHA1-Digest: iA2TluYMueZaOvIw+UZ0ElU7bVA=

Name: META-INF/androidx.arch.core_core-runtime.version
SHA1-Digest: OGGiGAP82euSpAMCew2iu3rdTeE=
```

Figure 4: MANIFEST.MF File Showing Hash Values of Listed Resources

- CERT.SF

  Not to complicate things or cause confusion, but this file is basically another hash digest file. Each entry in the MANIFEST.SF file has three lines, including the blank spaces in-between entries. That said, the CERT.SF file contains the hash of those three lines; that is, the Name:, SHA1-Digest:, and the third line that's blank.



```
Signature-Version: 1.0
Created-By: 1.0 (Android)
SHA1-Digest-Manifest: jwDvMnSrVaKJhZCfLunxQ+RsYA0=
X-Android-APK-Signed: 2

Name: AndroidManifest.xml
SHA1-Digest: NlzPpvtYWeANm0zAP6wAtNtIdYo=

Name: META-INF/androidx.appcompat_appcompat.version
SHA1-Digest: APIuYpbXlf2dFYms9cX8G5d7Ixk=

Name: META-INF/androidx.arch.core_core-runtime.version
SHA1-Digest: e1tut2kK2rB7RpspgrtY2fhXIus=
```

Figure 5: CERT.SF File Showing Hash Values of the Three-Lined Entries within the MANIFEST.MF

- CERT.RSA
  Here is where public and private keys are used for certificate signing. I'll expand a bit below but we can use this to see who signed the released APK.

The three of these got me a bit confused when I was first learning them and still only sort of understand them. I can only suggest reading this StackOverflow post or even this StackOverflow discussion. Where I got confused was trying to re-create each hash, which I failed even on my own test APK. In researching more, the explanation is beyond the scope of this article and I, again, will steer you towards the great StackOverflow. In short, it depends on which character set was used and how the original content was compiled in the first place.

How I understood the MANIFEST.MF file was: line one is the filename, line two is the Base64-encoded SHA1 hash of that file's content, line three is an intended blank line. Then the CERT.SF file somewhat

Privacy - Terms

repeats the idea but by naming the filenames and generating a different hash. This different SHA1 hash is, too, Base64-encoded, but is calculated from the three lines mentioned in the MANIFEST.SF file. CERT.RSA holds the organizations identity releasing said APK. If you see an app both on Google Play and an unknown third-party location, you can check this certificate and verify if there were changes to the app. An example of malicious intent would be altering a popular app from Google Play, signing it with a different certificate, then offering this APK on third party repositories. This is a prime example of why one shouldn't sideload apps.

Conclusion

To recap, APK is shorthand for *Android Package Kit* and is a file format used by electronic devices running the Android operating system. We're able to simply unzip an APK, as it's just an archive, but that's not an easy way to go about reverse engineering them as some files of interest are still in binary format. For the purposes of demonstration, however, I did simply unzip some sample APKs. There is a plethora of open source tools available for better reverse engineering APKs and I will write about them as I compare and contrast like tools.

The main files of interest are the AndroidManifest.xml and the classes.dex files. We'll get into these in future posts. Another focus point, from a security perspective, is the META-INF directory. This houses associated certificates used for signing and I will continue poking into this to better understand them myself. I also wanted to clarify that not all APKs will have just what was depicted in the above Figures. Some APKs had way more folders that require investigation to see what their purposes are.

Share This:

Related Posts

How to Learn Any Programming Language

Rogue Femtocell Sniffs Cellular Data - WSWiR Episode 70

Security Researcher Track: 102

Filed Under: Uncategorized

Tagged With: mobile security, reverse engineering

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Privacy - Terms

Name *

Email *

I have read and accept the WatchGuard Privacy Policy *

POST COMMENT

The 443 Podcast

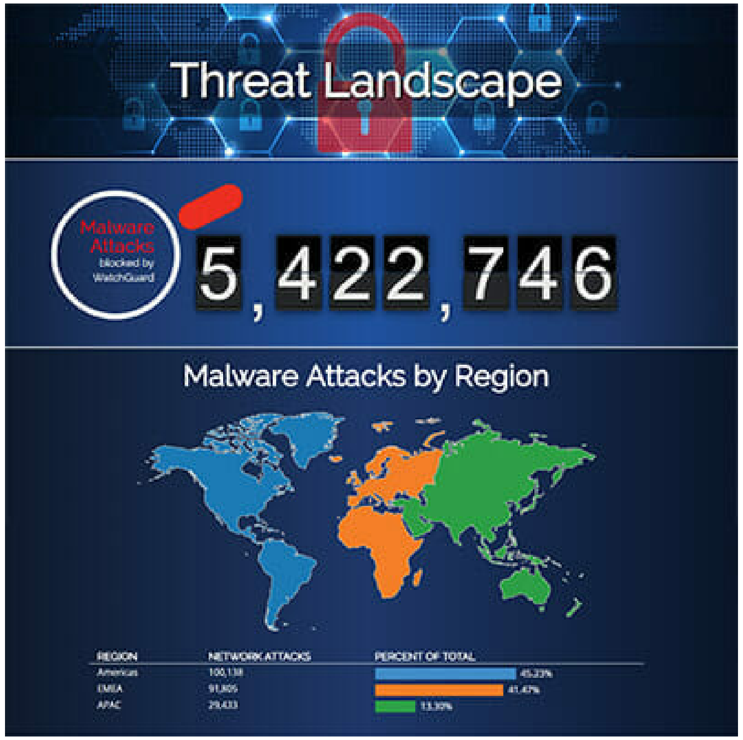
A weekly podcast featuring the leading white-hat hackers and security researchers. Listen Now

Privacy - Terms

Threat Landscape

Filter and view Firebox Feed data by type of attack, region, country, and date range. View Now

Privacy - Terms



REGION	NETWORK ATTACKS	PERCENT OF TOTAL
Americas	100,138	45.23%
EMEA	91,805	41.47%
APAC	29,433	13.30%

Top Posts

Penetration Testing

Hospital Horror Stories!

XKCD Forum Database Breached

9th Circuit Court of Appeals Makes CFAA Law More Confusing

Email Newsletter

Sign up to get the latest security news and threat analysis delivered straight to your inbox

Email Address

Daily ✓

Weekly

SUBSCRIBE

By signing up you agree to our Privacy Policy.

Privacy - Terms

Secplicity®
Security Simplified

STAY IN TOUCH

RECENT POSTS

Unpatched 0-Day Android Vulnerability

A Silent Mobile Threat: Simjacker

9th Circuit Court of Appeals Makes CFAA Law More Confusing

Android APK Reverse Engineering: What’s in an APK

Attackers DDoS Wikipedia, Twitch and World of Warcraft

View All

SEARCH

Privacy - Terms

Search the site ...

ARCHIVES

Select Month ▾