NYU | TANDON

Hidden Vulnerabilities in Google Play Store Apps

CS-GY – 9223 – Mobile Security

Michael Agee
maa9605@nyu.edu

Henry Post
hp2376@nyu.edu

04.14.24

Introduction

The Google Play Store is the official app store for devices running on the Android operating system. It serves as a centralized platform where users can discover, download, and manage various types of digital content, including apps, games, movies, TV shows, books, and music. Its most important elements with respect to security are:

- **Trust**: Google implements robust security measures to protect users and their devices from malware and other security threats. Apps undergo a review process before they are published on the Play Store, and Google continuously monitors and updates its security protocols to ensure a safe environment for users.

- **Updates**: The Play Store makes it easy for users to update their apps to the latest versions, ensuring they have access to new features, bug fixes, and security patches. Automatic updates can be enabled to streamline the process further.

Introduction

Security is paramount in the digital landscape due to the increasing reliance on technology in our daily lives. As we store sensitive information, conduct financial transactions, communicate, and access various services online, the need to safeguard our digital assets from threats such as cyberattacks, data breaches, malware, and other vulnerabilities becomes imperative.

Background

Several notable incidents and research findings have shed light on security vulnerabilities in Android apps over the years. Here are a few examples:

- Exodus Spyware (2019): Researchers uncovered the Exodus spyware in 2019, which was being distributed through malicious apps on the Google Play Store. The spyware was capable of stealing a wide range of sensitive information from infected devices, including call logs, SMS messages, contacts, and device location. The incident raised concerns about the effectiveness of Google's app review process and the need for improved detection mechanisms to identify and remove spyware-infected apps from the Play Store

- FakeApps Malware Campaign (2018): Researchers uncovered a widespread malware campaign on the Google Play Store dubbed "FakeApps" in 2018. The campaign involved hundreds of malicious apps masquerading as popular applications like WhatsApp and Minecraft. Once installed, these apps would display intrusive ads, redirect users to phishing websites, or even steal sensitive information from the device. The incident underscored the challenges of app vetting and the need for improved security measures to detect and prevent the distribution of malicious apps on the Play Store.

Background

Recent studies on the increases in secret exposure, show that despite the communication of industry best practices, exposed secrets in repositories and decompilable code are increasing.

- “A comparative study of software secrets reporting by secret detection tools,” IEEE Conference Publication | IEEE Xplore, Oct. 26, 2023. https://ieeexplore.ieee.org/abstract/document/10304853
- “Connecting the .dotfiles: Checked-In Secret Exposure with Extra (Lateral Movement) Steps,” IEEE Conference Publication | IEEE Xplore, May 01, 2023. https://ieeexplore.ieee.org/abstract/document/10174120

Methodology

App Selection and Tools Used

APK Selection

Our first task was to select apps from the Play Store that we deemed to be indicative of common apps. We divided our target apps into two groups, 10 most popular apps and a random selection of apps

- 10 Most Popular App:
  - Facebook, Messenger, Whatsapp, Instagram, TikTok, Facebook Lite, Telegram, SHAREit, Netflix, Snapchat
- Random Apps:
  - Student Portal, QstudentConnection, Axis Mobile, BHIM Axis Pay, PrismHR, ClientApp, Verizon Client, AppFolio, Paycom, InteliChart

APK Harvesting

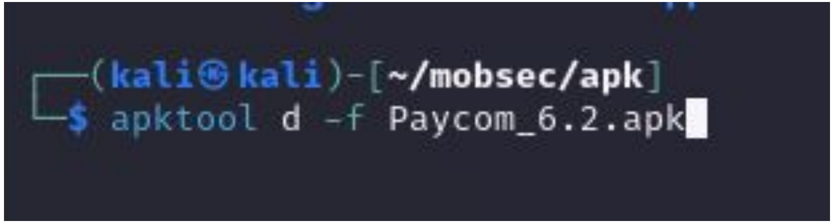
Direct downloading of APKs from the Play Store was not possible, so we had to use alternative means to harvest APK files. The use of free sites must be done carefully as apks can be refactored to include malware. We chose to use a website called "apkcombo.com".

- This tutorial was helpful:
  - How to Download an APK from the Google Play Store (howtogeek.com)
- APK Downloader
  - APK Downloader - Download APK & OBB (Latest Version) (apkcombo.com)

APK Unpacking

APK files are not much more than just zip files with the compressed binaries and support files in the package. We could simply unzip these files using built-in decompression tools in Kali. However, some data loss can occur with this method. To get a more ordered and human readable file set we use a tool called `apktool`, and later used `jadx`.

We used the following command to unpack the APKs:

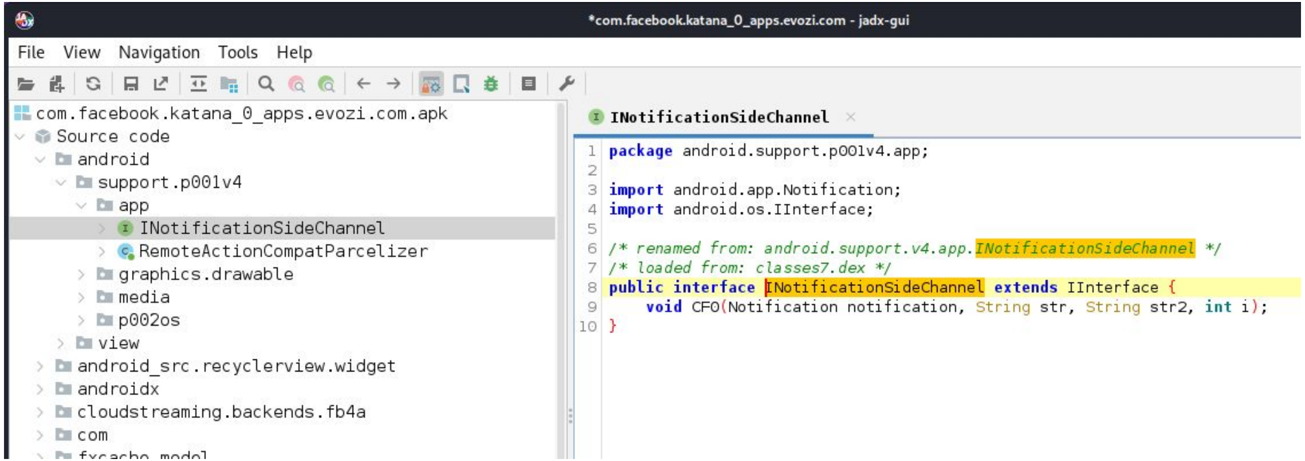


```
┌──(kali㉿kali)-[~/mobsec/apk]
└─$ apktool d -f Paycom_6.2.apk
```

APK Decompiling

Once we were able to unpack the APK file we needed a way to decompile the .DEX files for inspection. We were most successful with the `jadx` decompiler. It has a GUI version as well that was very helpful.

This tool needs to be configured to use more memory depending on the size of the DEX files.



```
com.facebook.katana_0_apps.evozi.com.apk
  Source code
    android
      support.p001v4
        app
          INotificationSideChannel
          RemoteActionCompatParcelizer
        graphics.drawable
        media
        p002os
      view
    android_src.recyclerview.widget
    androidx
    cloudstreaming.backends.fb4a
    com
    fxcache_model
```

```java
package android.support.p001v4.app;

import android.app.Notification;
import android.os.IInterface;

/* renamed from: android.support.v4.app.INotificationSideChannel */
/* loaded from: classes7.dex */
public interface INotificationSideChannel extends IInterface {
    void CF0(Notification notification, String str, String str2, int i);
}
```

Secrets and SAST Scanning

Scanning for secrets and SAST vulnerabilities source code can take hundreds of hours, if you're looking at 10,000 lines of code and multiple vulnerability types possibly hidden in those lines.

In order to get a comprehensive scan of the code, we used multiple scanning tools.

1. Uploaded source code to individual public git repositories,
2. Used Snyk, `gitleaks`, and Gitguardian tools to search for secrets and SAST vulnerabilities,
3. Used `apkurlgrep` to search for API URLs

SAST and Secrets Scans

Findings – Classification

Both of our scanning services categorized vulnerabilities and secrets in to 4 groups which we present here. While the scanners found many vulnerabilities we focused only on the “Critical” ones.

The criteria is outlined below:

Level	Description
Critical	May allow attackers to access sensitive data and run code on your application
High	May allow attackers to access sensitive data in your application
Medium	Under some conditions, may allow attackers to access sensitive data on your application
Low	Application may expose some data that allows vulnerability mapping, which can be used with other vulnerabilities to attack the application

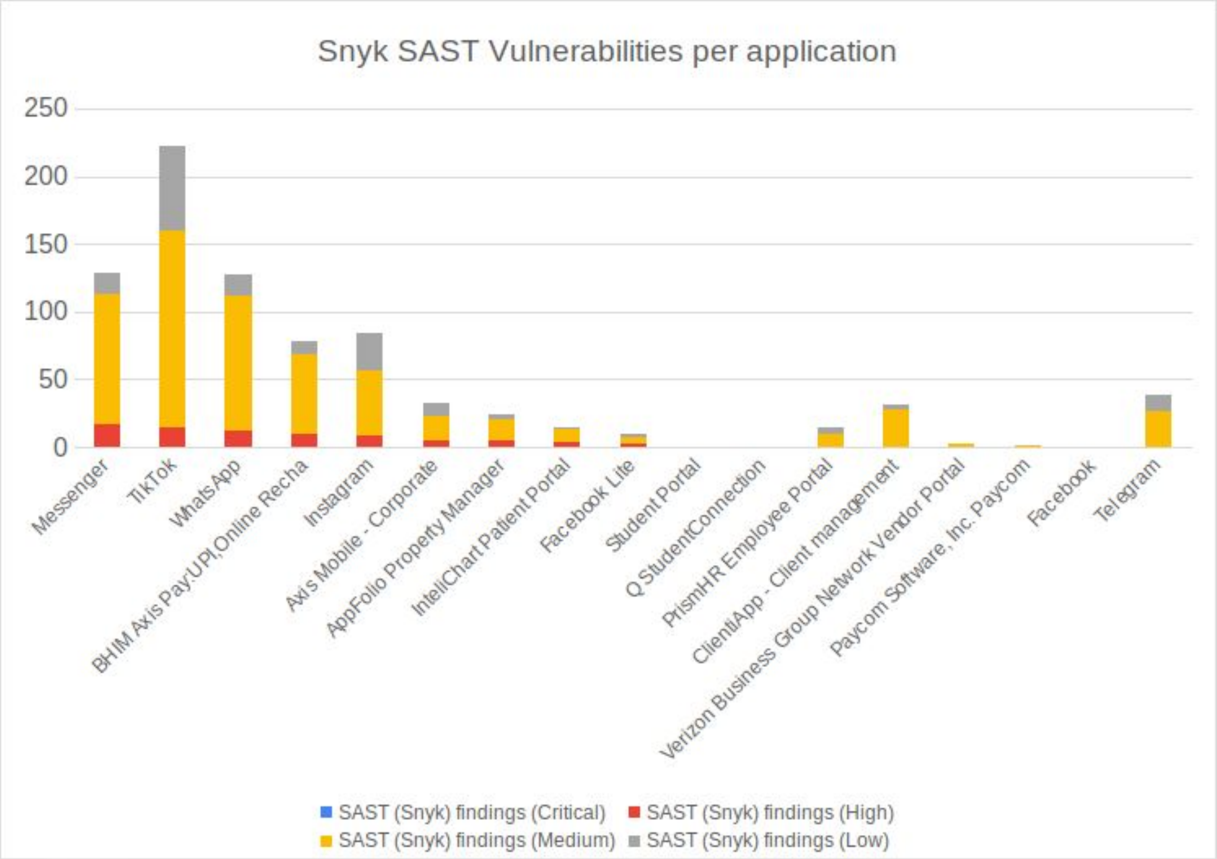
Findings – Distribution

The distribution of secrets found among the APKs can be seen on the right.

App Name	Secrets	Critical	High	Medium	Low
Student Portal	0	0	0	0	0
Q StudentConnection	0	0	0	0	0
Axis Mobile - Corporate	3	0	5	18	10
BHIM Axis Pay	0	0	10	59	9
PrismHR Employee Portal	0	0	0	10	5
ClientiApp - Client management	0	0	0	28	3
AppFolio Property Manager	0	0	5	15	4
InteliChart Patient Portal	0	0	4	9	1
Verizon Business Group Network Vendor Portal	0	0	0	2	0
Paycom Software, Inc. Paycom	0	0	0	1	0
WhatsApp	4	1	12	100	16
Facebook	16	4	0	0	0
Messenger	19	6	17	96	16
TikTok	22	1	15	145	62
Instagram	121	4	8	48	28
Facebook Lite	0	0	2	5	3
SHAREit	4	N/A	N/A	N/A	N/A
Netflix	0	N/A	N/A	N/A	N/A
Snapchat	1	N/A	N/A	N/A	N/A
Telegram	1	0	0	26	12

Findings – SAST Findings

Here you can see a distribution of SAST findings, by type, over each application.

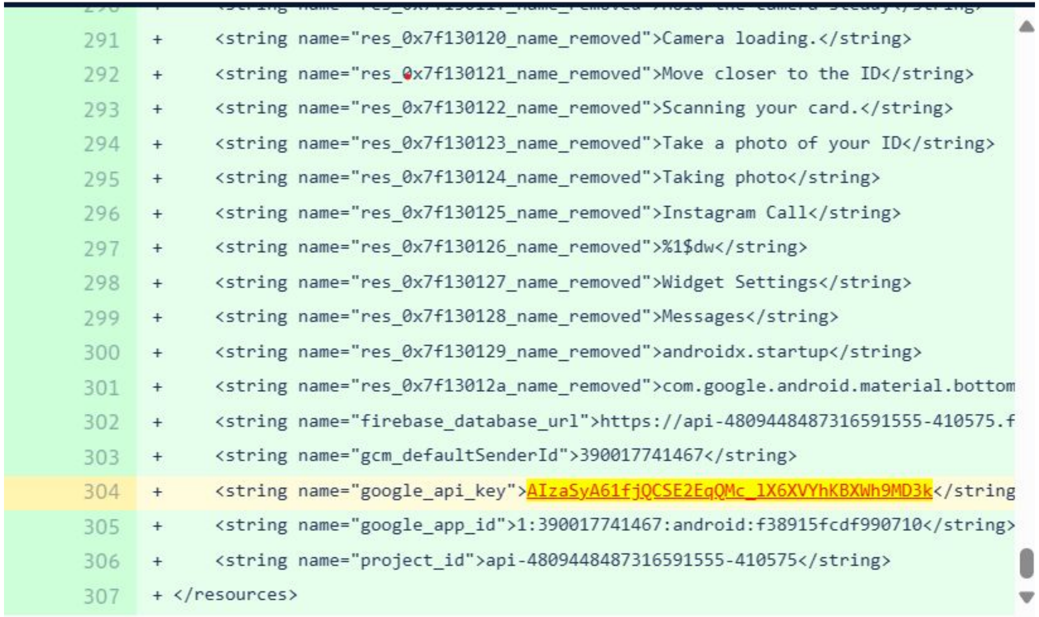


NYU | TANDON

Findings – Example of Critical Secret

A Google API key found. An attacker can use this to abuse the API by sending commands directly to it like the following string:

https://maps.googleapis.com/maps/api/geocode/json?address=ioki%20gmbh&key=[GOOGLE_API_KEY]

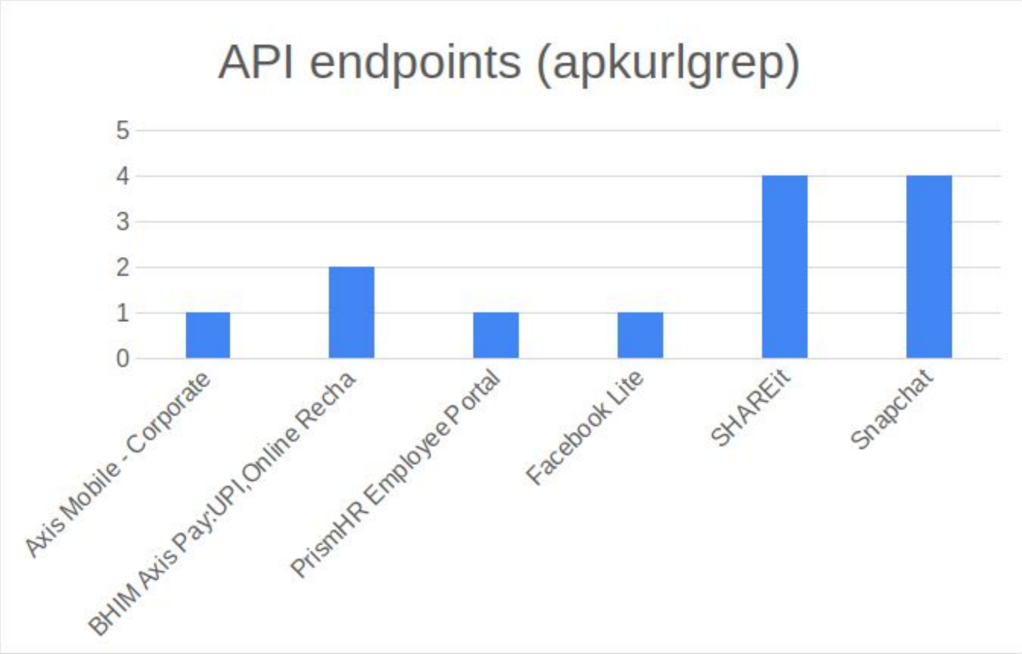


```
+     <string name="res_0x7f130120_name_removed">Camera loading.</string>
+     <string name="res_0x7f130121_name_removed">Move closer to the ID</string>
+     <string name="res_0x7f130122_name_removed">Scanning your card.</string>
+     <string name="res_0x7f130123_name_removed">Take a photo of your ID</string>
+     <string name="res_0x7f130124_name_removed">Taking photo</string>
+     <string name="res_0x7f130125_name_removed">Instagram Call</string>
+     <string name="res_0x7f130126_name_removed">%1$dw</string>
+     <string name="res_0x7f130127_name_removed">Widget Settings</string>
+     <string name="res_0x7f130128_name_removed">Messages</string>
+     <string name="res_0x7f130129_name_removed">androidx.startup</string>
+     <string name="res_0x7f13012a_name_removed">com.google.android.material.bottom
+     <string name="firebase_database_url">https://api-4809448487316591555-410575.f
+     <string name="gcm_defaultSenderId">390017741467</string>
+     <string name="google_api_key">AIzaSyA61fjQCSE2EqQMc_lX6XVYhKBXWh9MD3k</string
+     <string name="google_app_id">1:390017741467:android:f38915fcdf990710</string>
+     <string name="project_id">api-4809448487316591555-410575</string>
+ </resources>
```

Findings – API Endpoints

Here is a graph that shows embedded API URLs within APK files.

- Axis Mobile - Corporate
  - `idpm.axisbank.co.in` - Running an HTTP server
- BHIM Axis Pay:UPI, Online Recha
  - `[upiuat.axisbank.co.in/v1/](upiuat.axisbank.co.in/v1/)`
    - /v1/bank/transactions/pay
    - ...etc
- PrismHR Employee Portal
  - Found 1 URL: https://epycorp-ep.prismhr.com/apis/ep/peos?fwdClientCode=
    - Seems to be used with a URL parameter. We could probably fuzz the fwdClientCode parameter.



Discussion:

Can Android APK Secrets Be Made Safe?

Prior Research

- Our finding seemed to coincide and reinforce the findings of some earlier research would took a much wider sample of git repositories. In a paper from 2022 paper[1], researchers found nearly 200,000 API keys and tokens checked into the repositories.
- However, our research focuses on decompiled binaries. Recent research suggest that this is no safer than a private repository unless secure coding techniques are used and enforced. [2]

Challenges Of Mitigation

There are few good alternatives with when trying to secure secrets in your code, they have all have their costs:

- Encryption – Encryption has a cost in processor cycles and a cost of complexity to implement.
- Limiting Resources – We can limit apps to only retrieve secrets when on trusted networks, however, this limits functionality.
- Additional Authentication – This could mean MFA or the user provides additional credentials to verify the requesting apps is legitimate.

Best Practices

Current research suggest the some best practices that may mitigate secret exposure in binaries:

- Server-side Implementation – Don’t allow mobile apps to directly connect to 3rd party apps.
- Short-Lived Secrets – Rotate keys frequently
- External Secret Management – Store secrets in an external, secure secrets vault - i.e. Hashicorp Vault, CyberArk, etc.

Conclusions

Conclusion

Despite numerous studies and efforts to address the issue, exposed secrets in source code repositories persist. Our research has identified several factors contributing to this trend:

- Developer education and awareness are crucial
- Insufficient code reviews (**especially in legacy codebases**)
- Compliance with security standards
- Continuous monitoring and scanning of code repositories

Conclusion – Future Mitigations

Future mitigations might include:

- Increased Regulation
- More expensive liability and cybersecurity insurance to non-compliant developers
- Third party code repository monitors
- Enforced SAST/Secrets code scanning of all Android applications, before being released to the Google Play store.

Conclusion – Additional Research

From these conclusions we can see there needs to be some additional root cause analysis of the issue.   Possible research topics could include:

- Management of KPIs in Software Development
- Executive Policy in Secrets Management
- Developer Understanding of Secrets Risk
- An automated workflow for scanning, review, and remediation

Conclusion – Additional Work

- Our processes were somewhat manual in this research project...
- In the future, these processes could be automated and chained together:
  - Downloading APK files
  - Decompiling it with `jadx-gui`
  - Running Snyk, gitleaks, apkurlgrep on the source code
  - Run `nmap` on the API URLs
  - Run pentesting tools, DAST scanners on the API URLs

Citations

[1] "Security Analysis on Android Application Through Penetration Testing using Reverse Engineering," IEEE Conference Publication | IEEE Xplore, Mar. 01, 2023. https://ieeexplore.ieee.org/abstract/document/10127653

[2] "Detecting and mitigating Secret-Key leaks in source code repositories," IEEE Conference Publication | IEEE Xplore, May 01, 2015. https://ieeexplore.ieee.org/abstract/document/7180102

[3] "What are the Practices for Secret Management in Software Artifacts?," IEEE Conference Publication | IEEE Xplore, Oct. 01, 2022. https://ieeexplore.ieee.org/abstract/document/9973029/author

Source code/workflow

See https://github.com/meltingscales/NYU-CS-GY-6xxx-mobile-security-final-project for:

- Source code, scripts
- Workflow for decompiling and scanning APKs
- Individual APK files
- Breakdown of API URLs
- Breakdown of secrets findings
- Breakdown of SAST findings

Thank you!