MobSec Final Project Proposal
APK Dangers

Unveiling the Hidden Vulnerabilities of Play Store Apps

Michael Agee | Henry Post

Intro

The Google Play Store, having millions of applications across diverse categories, serves as a gateway for Android users to access a vast digital landscape. However, this immense app ecosystem harbors a discrete vulnerability: the potential for security flaws that threaten user privacy, data integrity, and even device functionality. This paper investigates a selection of Play Store apps and scans them for vulnerabilities, exploring their nature, impact, and potential mitigation strategies.

Background

The security of publicly-available android apps has always been an important one. Often times, companies will build APIs or internal systems that only communicate with android apps, and they implicitly trust the app to not do anything malicious. Or, alternatively, companies will public android applications that have security vulnerabilities within them. Therefore, it's important to perform code scanning and security testing on android applications.

The current state-of-the-art solution to vulnerabilities in android applications is to run SAST (static appsec testing) and SCA (software composition analysis) tools against the source code of the android application. This is performed in this paper, and we also add other analysis tools: Secrets scanning (via gitleaks), a process by which credentials can be detected in source code, is used. In addition to this, we use a tool called "apkurlgrep" to extract URLs from the android application. These internal URLs can later be probed or tested to get more information about the backend of the android application.

Scope

Our investigation stems from a growing concern about the rising number of security incidents linked to Play Store apps. We aim to shed light on the possible risks associated with Play Store apps and contribute to the ongoing discussion around user protection and app security best practices.

Methodology

We will determine this by random sampling of 10 APKs, as well as picking an additional 10 of the most popular APKs. We will use "apktool" and "jadx" for decompiling APKs, we will use "apkurlgrep" to find URLs within APKs, we will use "Snyk" to do SAST (Static AppSec Testing) scanning, and we will use "gitLeaks" for secrets scanning.

Key Areas

Types of vulnerabilities

Examining common vulnerabilities like insecure coding practices, permission overreach, and lack of encryption.

Impact on users

Assessing the potential harm caused by vulnerabilities, including data theft, financial losses, and identity compromise.

Root causes

Identifying the underlying factors contributing to app vulnerabilities, such as developer oversight, inadequate testing, and outdated libraries.

Mitigation strategies

Exploring potential solutions, including improved developer education, stricter app review processes, and user awareness campaign

Data

The data from our investigation of publicly-available APKs can be found below.

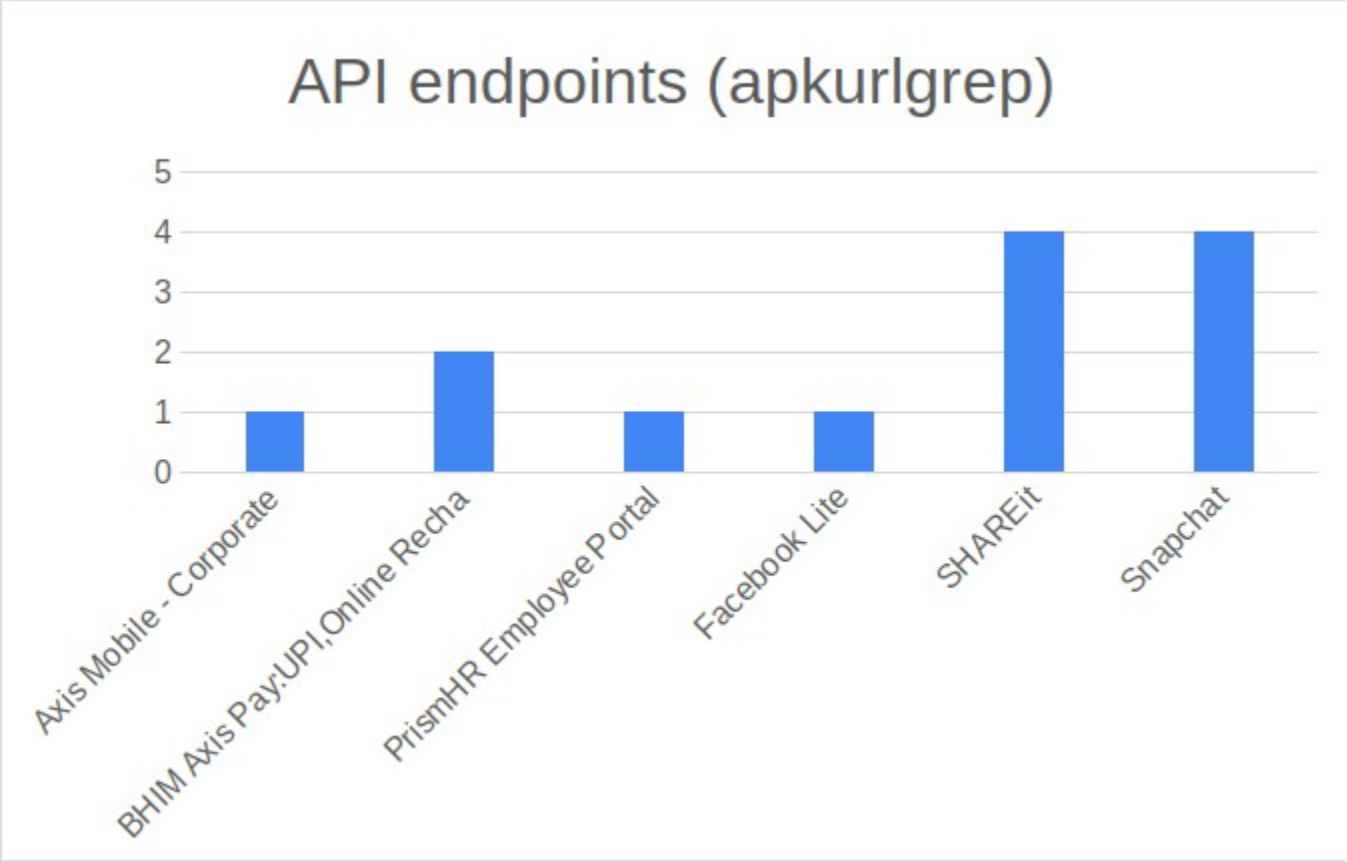
Gitleaks

The above figure shows that a decent number of hardcoded secrets were found within messaging apps. These could be API keys, passwords, or other credentials. From the surveyed applications, this is a breakdown of the categories of exposed credentials:

- Axis Mobile - Corporate:
    - 2 Generic API keys
    - 1 Google Maps API key
- WhatsApp
    - 1 Google Maps API key
- Facebook
    - 1 Google API key
    - 1 Facebook App key
- Messenger
    - 1 Facebook App key
- TikTok
    - 1 Google Maps API key
- Instagram
    - 1 Google API key
    - 1 Facebook App key

It seems that most of the API keys are google/google maps. A tool that automatically tests these hardcoded API keys would be very useful, as it would prevent API keys from leaking on the Google Play store.

apkurlgrep

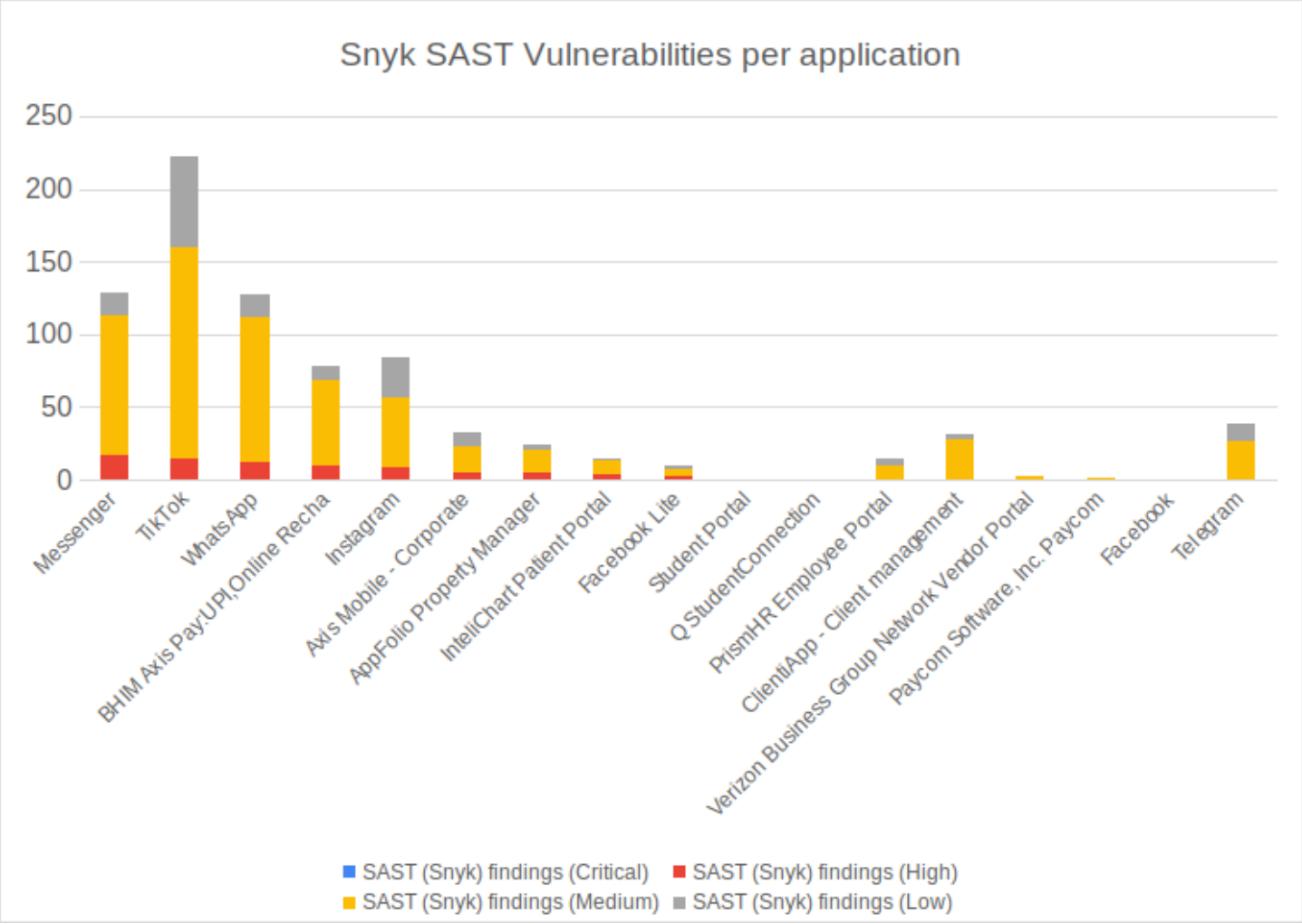


The above figure shows API endpoints that were found by an API URL scanning tool, “apkurlgrep”. Below is a breakdown of the types of API routes per application.

- 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
        - /v1/customer/accounts
        - /v1/customer/accounts/mobreg
        - ...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.

It would also be very, very useful to create a tool that automatically searched public APKs for API endpoints and performed an automated penetration test on them.

Snyk SAST



As for the SAST findings, I will summarize them.

Many software applications on this list use outdated encryption algorithms. Some use DESede, some use AES with insecure cipher modes like CBC, some have hardcoded IVs, some seed the encryption with `Random.nextInt`, etc. Only 2 applications have SQL Injection vulnerabilities.

The trend here is encryption vulnerabilities. Software applications should never do the above — using old encryption algorithms, insecurely seeding RND, and using insecure encryption modes.

Conclusions

There is a plethora of useful and interesting data stored in APK files in Google Play. You can find hardcoded API keys, software vulnerabilities, internal API endpoints, and other vulnerabilities.

Future work ideas: Automate the process

This manual work — downloading APKs, decompiling them, inspecting them with tools — can and should be automated. This could potentially be very useful future work. The flow for this would be as follows:

1. Input: Download an APK file

2. Work: Decompile it with `jadx-gui`
    a. Output: Source code.

3. Work: Run Snyk SAST on the source code.
    a. Output: SAST findings.

4. Work: Scan the source code with `gitleaks`
    a. Output: Exposed credentials.

5. Work: Run `apkurlgrep` on the APK file.
    a. Output: API URLs.

    Additionally...

6. Work: Run `nmap` on the API URLs:
    a. Output: Web services that are publicly exposed

7. Work: Run `dirb` on the API URLs:
    a. Output: Web directories that are publicly available

8. Work: Run a DAST scanning tool on any API URLs:
    a. Output: DAST vulnerabilities.

References

[1] *Android Applications Pentesting*. Hacktricks. (n.d.). https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting/
[2] *GitLeaks.* GitLeaks. (n.d.). https://github.com/gitleaks/gitleaks
[3] *apkurlgrep.* apkurlgrep. (n.d.). https://github.com/ndelphit/apkurlgrep