Dataset Collection: We collected the certified version and latest version of the certified apps and only the latest versions of non-certified apps using APKPure. All of our collected apks can be accessed here: https://drive.google.com/file/d/1pkobSK5ba0EIIQ1H9rYIgBmHYMm9vMKN/view?usp=sharing

Static analysis:

1. To do the static analysis of the apks, we used jadx-gui and it can be downloaded from https://github.com/skylot/jadx/
2. We only focused on the base package of the apps mentioned in androidmanifest.xml file of an apk
3. We then searched specific terms in the code to find out if it has been used or not.
4. Our search term included these bag of words
    a. Cipher.getInstance
    b. "AES"
    c. "AES/ECB/NoPadding"
    d. "AES/ECB/
    e. AES/CBC/PKCS5Padding
    f. X509TrustManager
    g. X509ExtendedTrustManager
    h. checkServerAccepted
    i. checkServerTrusted
    j. getAcceptedIssuers
    k. HostnameVerifier
    l. SSL
    m. TLS
    n. SSLContext.getInstance
    o. PBEKeySpec
    p. MessageDigest.getInstance
    q. "MD5"
    r. "SHA-1" or "SHA1"
5. If any match was found, we then check if the code is in live upto 4 hops or not
6. We also checked the context of the code to figure out whether it can be exploited or not

Automated Tools:

We used three automated tools in this project to find out vulnerabilities. They are Cryptoguard, MobSF, APKHunt.

Cryptoguard:

1. The version we used for our analysis can be found here https://drive.google.com/drive/folders/1F6ApYY721nlRNKlz72DOeyjlhVEp32Uc?usp=sharing

2. After that,we ran the following commands to run the cryptoguard against our apks.

3. java -jar main/build/libs/main.jar "apk" <apk_path> "" 1

MobSF:

1. To run mobsf, we needed to install docker on our machine. We downloaded docker desktop from https://www.docker.com/get-started/.
2. Then we ran the following commands:
    a. docker pull opensecurity/mobile-security-framework-mobsf:latest
    b. docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
    c. Username: mobsf, password: mobsf

ApkHunt:

1. We already had installed git, golang, jadx, dex2jar
2. To run apkhunt, we cloned the repository from https://github.com/Cyber-Buddy/APKHunt/tree/main
3. cd APKHunt
4. go run apkhunt.go

Permission Analysis:

1. We installed all the certified apps on a testing Android device
2. We ran “adb dumpsys <base_package_name_of_apk>” to get the runtime permission requests
3. We compared the output against data collections mentioned in apps privacy policy and description

Data leak analysis:

We used flowdroid to analyze sensitive data leaks.

1. We downloaded flowdroid from here: https://github.com/secure-software-engineering/FlowDroid
2. Then ran the following command:

```
java -Xmx16g -jar <jar_file_path>\
  -a <apk_path>\
  -p /<platforms_directory_path> \
  -cg CHA \
  -s <our_source_sink_list> \
  -d \
  -sp \
  -nt \
  -o <output_xml>\
```

> /<output_text_file>2>&1

3. After getting leaks from Flowdroid, we manually validated their credibility
4. We checked the source and sink to learn whether they sensitive information or not

Vulnerability Exploitation:

1. To install mitmproxy, we ran “brew install mitmproxy”.
2. Then we ran “mitmweb” in the terminal to run the ui version
3. We installed our targeted vulnerable app in it
4. We changed the device proxy to manual and set ip address of our machine as proxy host. Also provided the corresponding port number where mitmweb was running in the network configuration.
5. We then initiated a login request and all the incoming network requests from our android device was interceptible in mitmweb

Vulnerability Mapping:

1. After getting all the vulnerabilities, we tried to find test rules of MASA (both old and latest versions) which talk about finding them
2. We found the previous version(1.5) of the MASA test guideline here: https://web.archive.org/web/20230208203056/https://mas.owasp.org/MASVS
3. Also, the newer version is available here: https://mas.owasp.org/checklists/
4. As newer version has more test cases, it was apparently easy for us to map the vulnerabilities to the corresponding test case