External Application Penetration Testing Technical Report

For

King Saud University

Document Info

Item	Description
Document Title	Mobile Application penetration testing
Requestor	King Saud University | College of Computer and Information Sciences | Information Technology department | IT 371: Application security | Teacher Manal AlAqeel
Author/s	Reema AlKraidees ,Reema Altayash, Reema Alowerdi, Layan Alzahrani
Date Created	12 April, 2022

Table of Content

1 Executive Summary

1.1 *Introduction*

Penetration testing involves simulating a malicious attacker's attack on a computer system, network, or application in order to assess the network's, application's, or application's security to uncover flaws.

Conducting a penetration test without knowing anything about the system being evaluated is known as black box penetration testing.

In this project, we will do black box mobile application penetration testing on the MPT2.apk file using tools like mobSF, jadx, genymotion, and ADB.

1.2 *Scope*

The specific scope of this project includes performing the Application Penetration test for the specified duration on the below mentioned applications.

To perform black box penetration testing for the MPT2.apk file we used the following tools:

- **Genymotion**: is an Android emulator which includes a complete set of sensors and features in order to interact with a virtual Android environment, which was helpful since we didn't have an actual android phone on hand.
- **Jadx**: is an unmaintained decompiler for the Java programming language, this tool helped us in producing source code.
- **mobSF**: Mobile Security Framework is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis, which helped us by pointing out the vulnerabilities.
- **Adb**: android debug bridge is a programming tool used for the debugging of Android-based devices, and we used it to control the Genymotion android device simulator.

Application Name	Platform	Version	Environment	Approach
PT	**Android**	**1.0**	**Windows**	**black box penetration testing**

1.3 *Risk Rating*

The risk rating for the issues and their impact on the operation of the organization is explained in the table 1 below. The overall risk rating reported will be based on vulnerability identification with its potential to be exploited by adversaries.

In general, the following factors were considered to arrive at the risk rating for vulnerability:

- Technical Impact: The extent to which an attacker may gain access to a system and the severity of it on the application. This metric will take the security triad CIA (Confidentiality, Integrity and Availability) values into account.
- Likelihood: This metric will take the Popularity and Simplicity of an exploit into consideration.
    - Popularity describes the existing or potential frequency of exploitation of the vulnerability.
    - Simplicity is the amount of effort required to exploit the vulnerability.

Overall Risk Severity				
Technical Impact (Confidentiality, Integrity, Availability)	HIGH	MEDIUM	HIGH	CRITICAL
	MEDIUM	LOW	MEDIUM	HIGH
	LOW	INFO	LOW	MEDIUM
		LOW	MEDIUM	HIGH
		Likelihood (Popularity and Simplicity)		

Table 1 Risk Severity

1.4 *Threat Security Level*

Vulnerabilities are categorized as **Critical**, **High**, **Medium**, **Low** and **Informational**.

Critical: Severe Impact on the affected application. They require immediate attention and resolution. Successful exploitation may provide the attacker **access to critical data**.

High: Severe Impact on the affected application. They require immediate attention. They are relatively easy for attackers to exploit and may provide them with **full control of the affected application.**

Medium: Moderate impact on the affected application. They are often **harder to exploit** and may not provide the same access to affected application.

Low: Limited impact on the affected application. They provide information to attackers that may assist them in mounting **subsequent attacks on the affected applications.** These should also be fixed in a timely manner, but are not as urgent as the other vulnerabilities.

Informational: It exposes information that target stake holders simply need to be aware of. These are for findings that are very difficult to exploit in practice.

1.5 *Summary Table*

The table below shows the summary of vulnerabilities disclosed during the Penetration Testing.

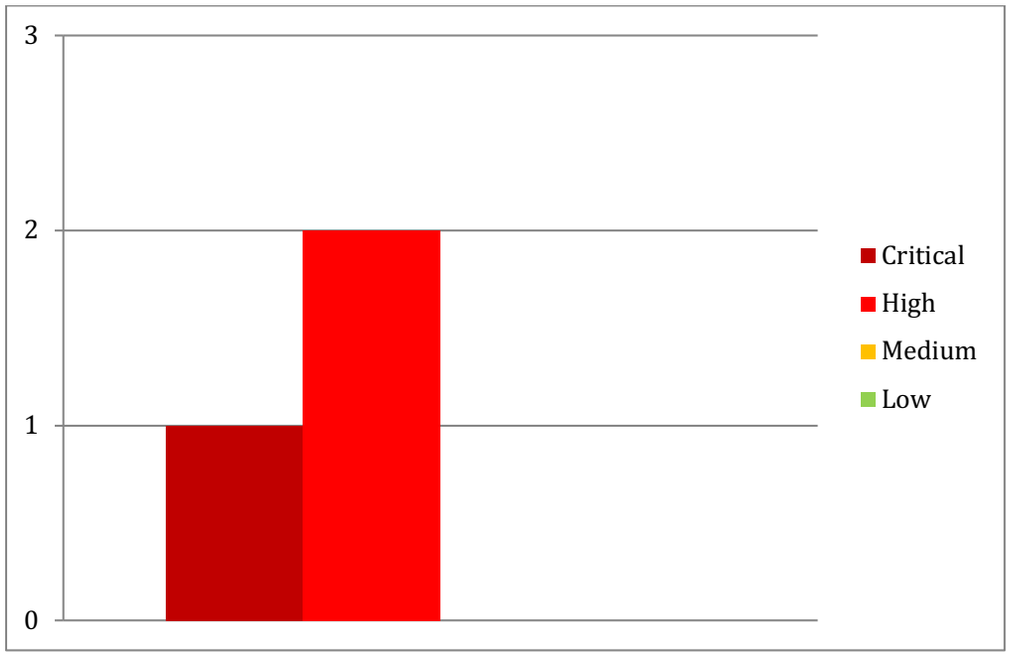
Mobile Application Penetration Testing

Critical	High	Medium	Low
1	2	-	-

1.6 *Summary Graph*

The following bar graph highlights the total number of vulnerabilities discovered during the penetration testing.

Figure 1 Application Penetration Testing



1.7 *Key Findings*

No.	Vulnerabilities Discovered	Platform	Severity Level
1	**Hardcoding sensitive data in source code. (sensitive data in plain text)**	Android	Critical
2	**Insecure logs. (unencrypted and visible data)**	Android	High
3	**Improper authorization. (activity is accessible by other apps)**	Android	High

2 Conclusion

During the penetration testing process, we discovered three vulnerabilities ranging from high to crucial. To strengthen the application's security, it is recommended to prioritize and adhere to the following advice on how to mitigate these vulnerabilities, so that an attacker cannot simply exploit or take advantage of those vulnerabilities.

- **Hardcoding sensitive data in source code:** hard-coded passwords make it easier for an attacker to guess the password. Its security level is critical because we have access to sensitive information, and we strongly advise you to fix it right now.

  Addressing the threat through mitigation: by keeping passwords out of the code, in a password- protected, encrypted configuration file or database.

- **Insecure Logs:** Revealing sensitive information in logs. An attacker can take advantage of this vulnerability and obtain sensitive information. Its security level is high, and you should try to fix this vulnerability immediately by following the following suggestion.

  Addressing the threat through mitigation: do not give access to the logs to unauthorized users and uses a good hashing algorithm to hash the passwords so that they are not stored as plain text. Therefore, if an attacker can access the logs, password hashing adds another layer of security.

- **Improper authorization:** The application does not perform an authorization check when trying to access information or perform an action. Its security level is high because the attacker will have control over the application, so it is important to try to fix this vulnerability immediately.

  Addressing the threat through mitigation: make sure to perform access control checks and grant the necessary privileges without the risk of privilege escalation.

3 Detailed Findings

3.1 *Vulnerability Table*

Severity Level	Risk Severity Level of the Vulnerability				
Technical Impact	Impact level	**Likelihood**		Popularity and Simplicity	
		Popularity	Popularity of the Exploit	**Simplicity**	Simplicity to Exploit
Observation	Our observation and description about the vulnerability				
Impact	Describes the possible Technical Impact if the vulnerability is Exploited				
Proof of Concept					
Evidence to support the finding					
Recommendation	High Level recommendation to mitigate the vulnerability				
OWASP Reference	OWASP Reference				
Reference	Reference related to the vulnerability				

3.2 *Limitations*

The limitations we faced were the following:

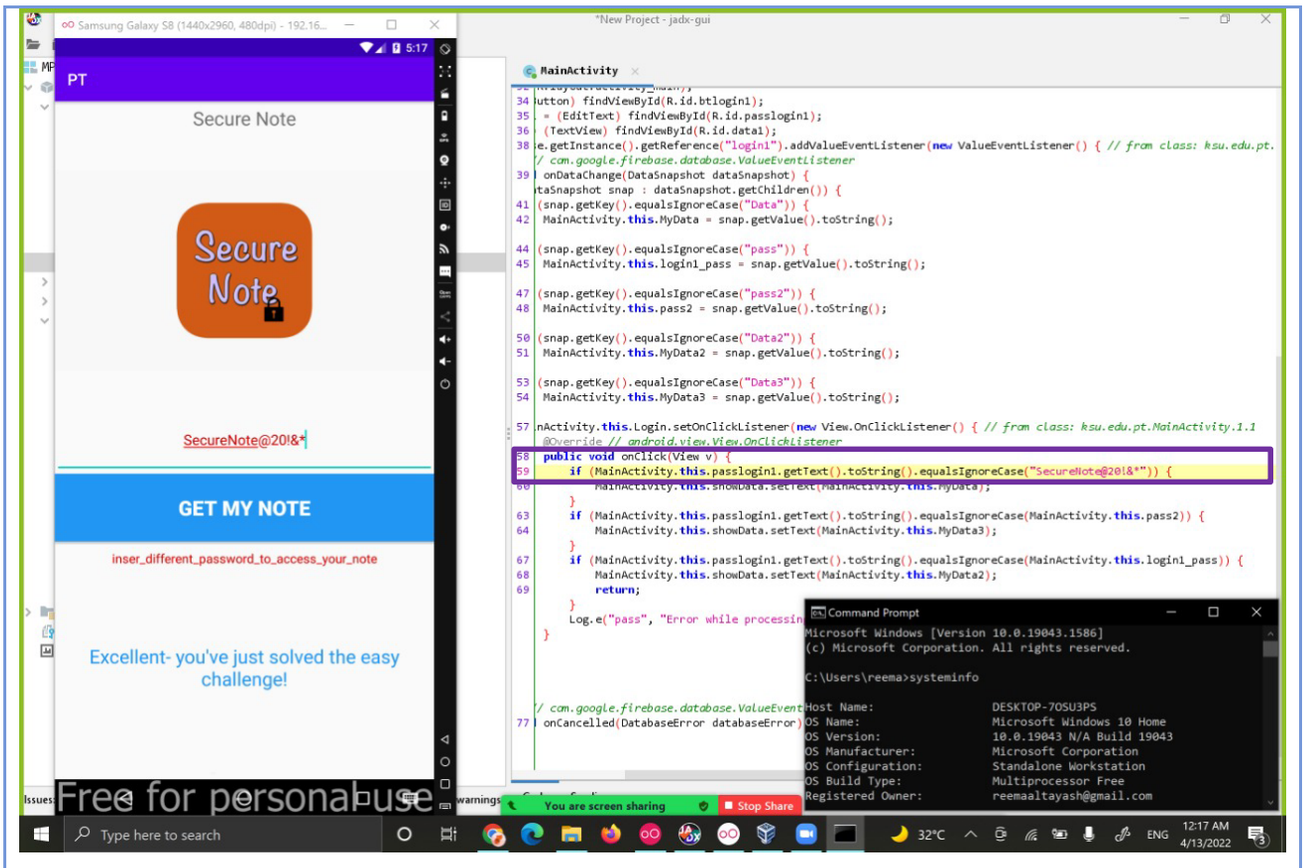
- Installing all the tools we needed was time consuming and also took a lot of memory space.
- mobSF needed a lot of requirements (python,jdk,GIT,etc..)
- The Genymotion sometimes froze and we needed to restart from the beginning of every run it froze in.
- Jadx refused to run on one of our devices and we couldn't figure out why, and we had to install it on another device.

3.3 *Technical Description of Findings*

This section explains the details of the identified vulnerability along with technical impact, Proof of Concept, recommendations and references related to the vulnerability.

3.3.1 Hardcoding sensitive data in source code

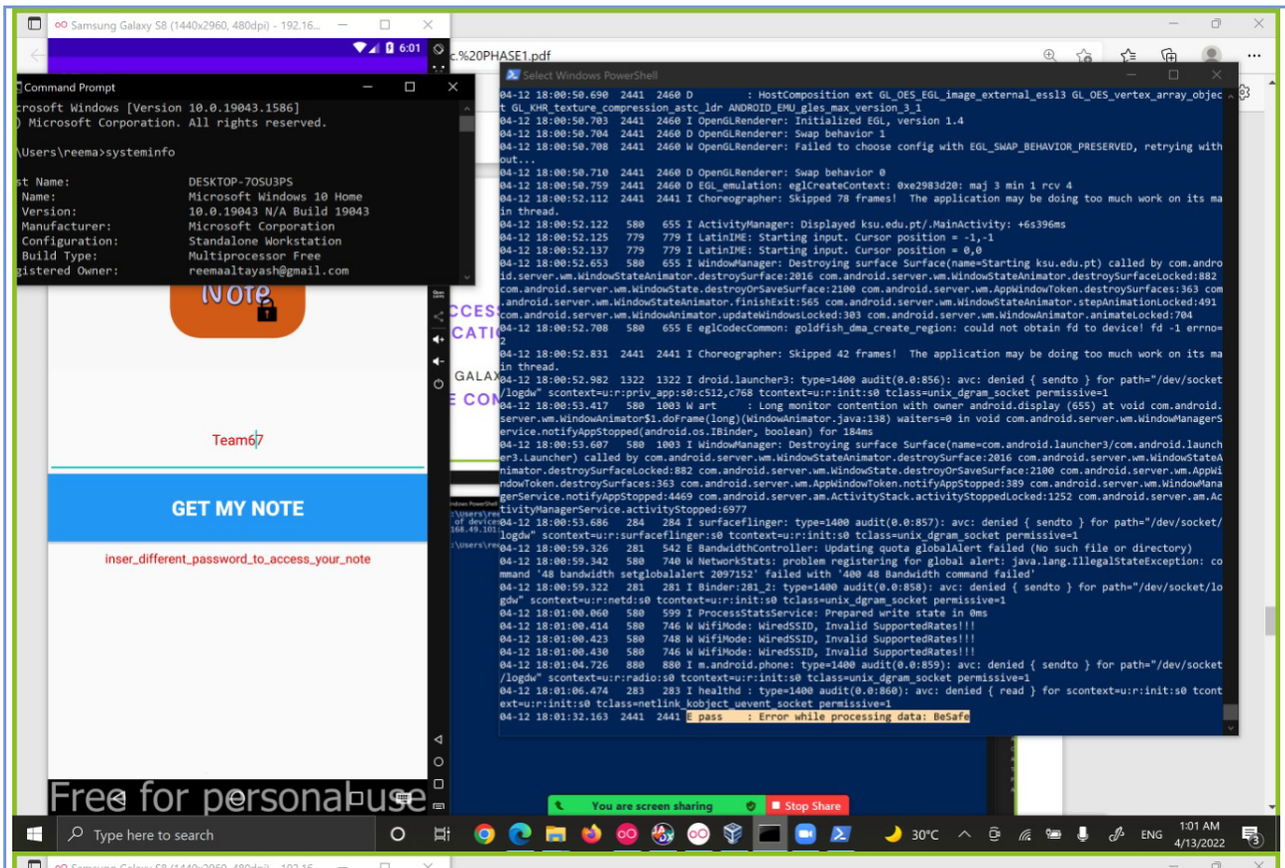
Severity Level	**CRITICAL**				
Technical Impact	HIGH	**Likelihood**		HIGH	
		Popularity	HIGH	**Simplicity**	LOW
Observation	We noticed that while uploading the MPT2.apk file in jadx, we were able to find the password easily because it was written in plain text in the source code of the login button main activity. This makes the application open to any attack.				
Impact	Many tools, such as jadx and Android Studio, make it easy to open the source code of any application, leaving your application vulnerable to attacks. As with this downloaded application, the source code is clearly visible, allowing attackers to read sensitive data and invade privacy. An attacker could also enter a password stored as plain text in the source code to log into an account. This violates application authentication.				
Platform	**Android**				
Proof of Concept					

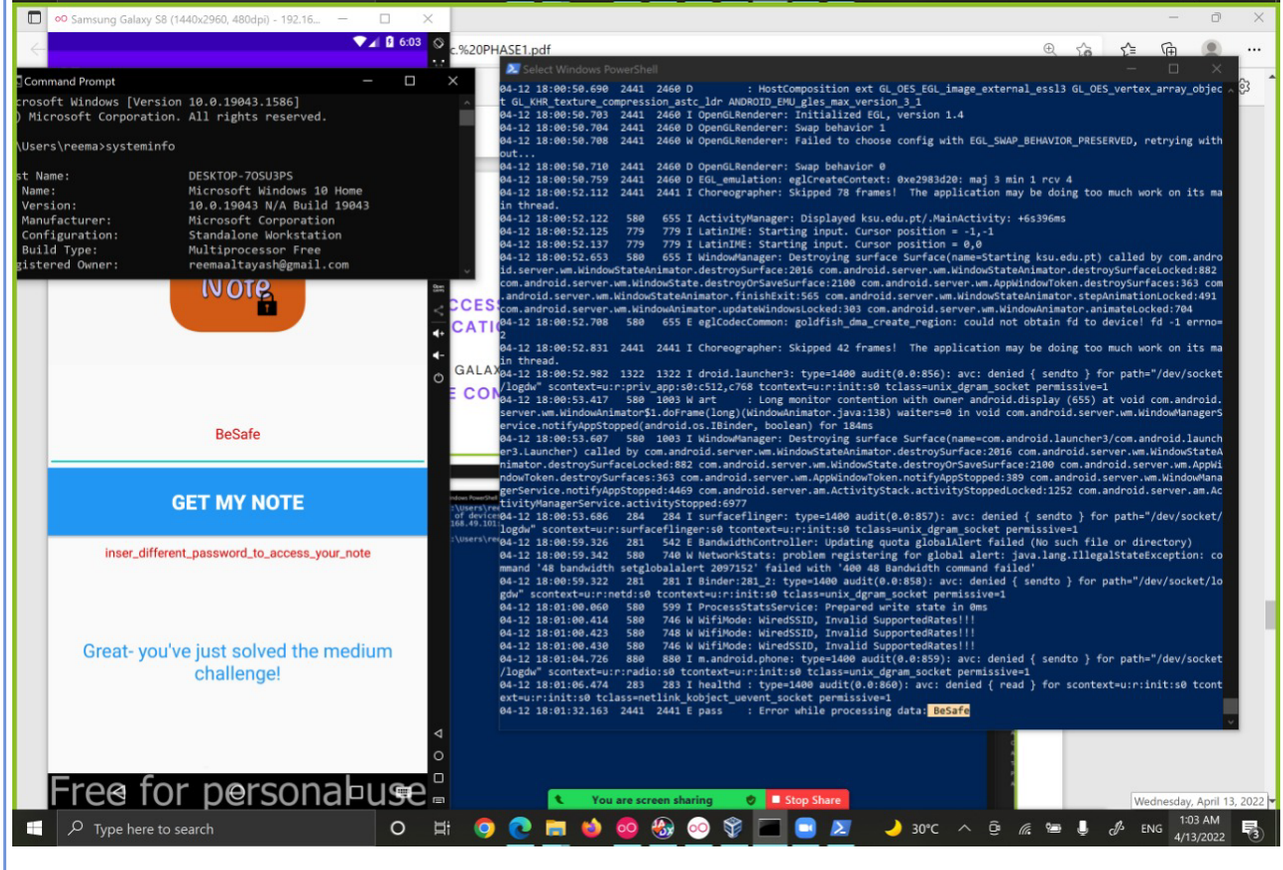


Recommendation	To prevent hackers from accessing plain text passwords is to not include the password in the source code. And, store hashed passwords outside of the code in a configuration file or a highly encrypted and protected database.
Reference	https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password

3.3.2 Insecure logs

Severity Level	**HIGH**				
Technical Impact	HIGH	**Technical Impact**		MEDIUM	
		Popularity	MEDIUM	**Simplicity**	MEDIUM
Observation	We used adb to check the log, then we experimented and entered the wrong password and checked the logs to find the correct password. We uploaded the file and found about this vulnerability in mobsF's manifest analysis of the code. When we click on the password it moves to the source code. And after reviewing it, we observed that the password was displayed in plain text when it shows an error in the log.				
Impact	Logging sensitive user data often gives attackers an additional, less protected path to gather information. In this case, an attacker could access the accounts with the password shown in the authentication breach log and read sensitive information in violation of privacy.				
Platform	**Android**				
Proof of Concept					

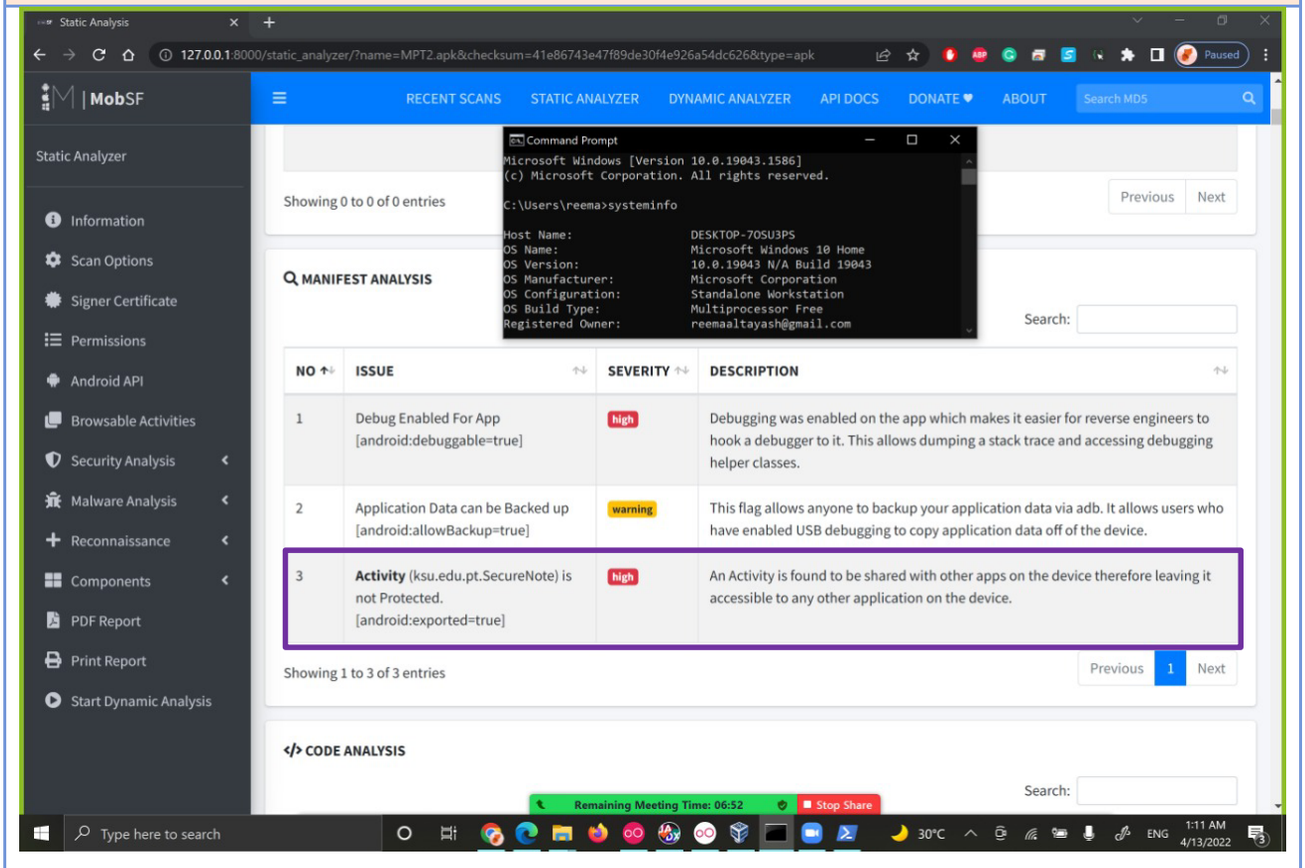


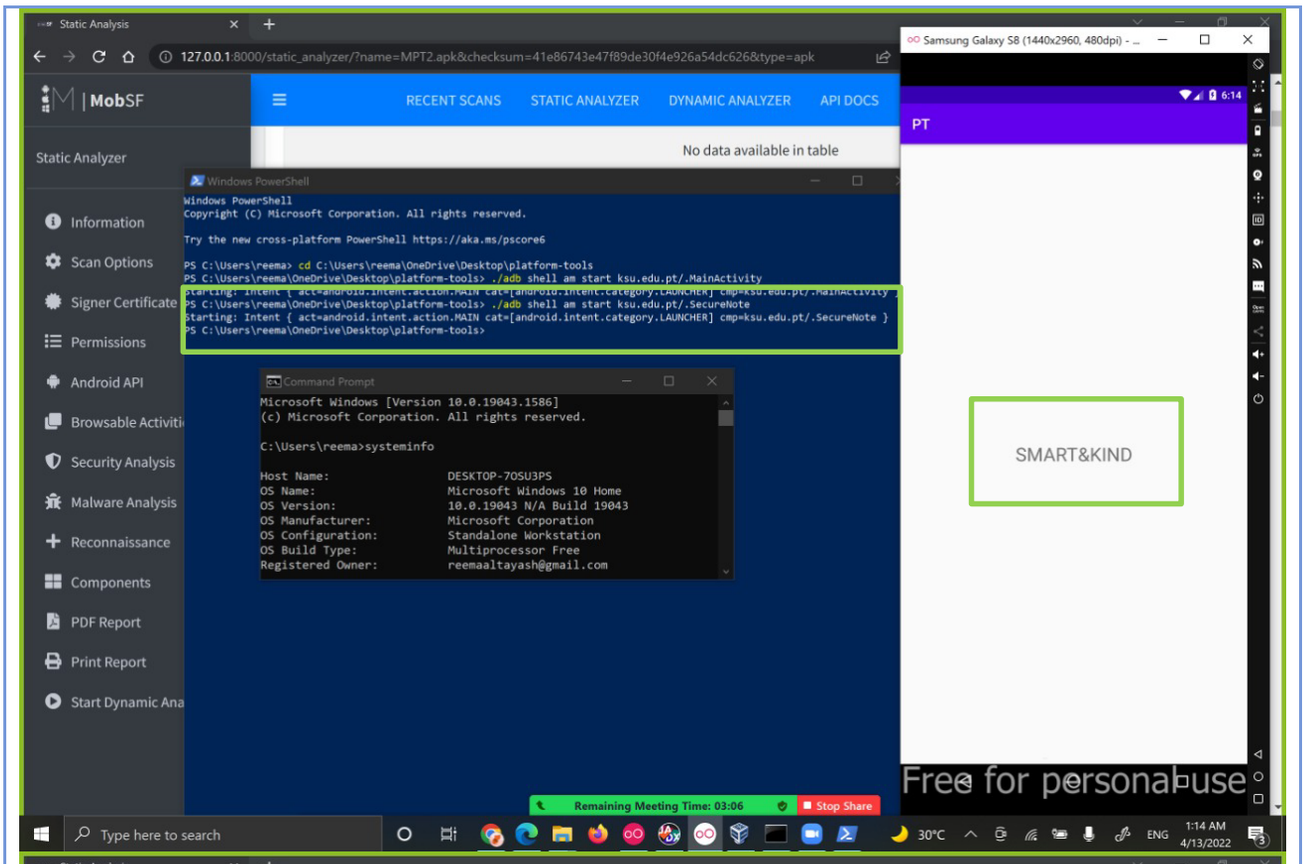


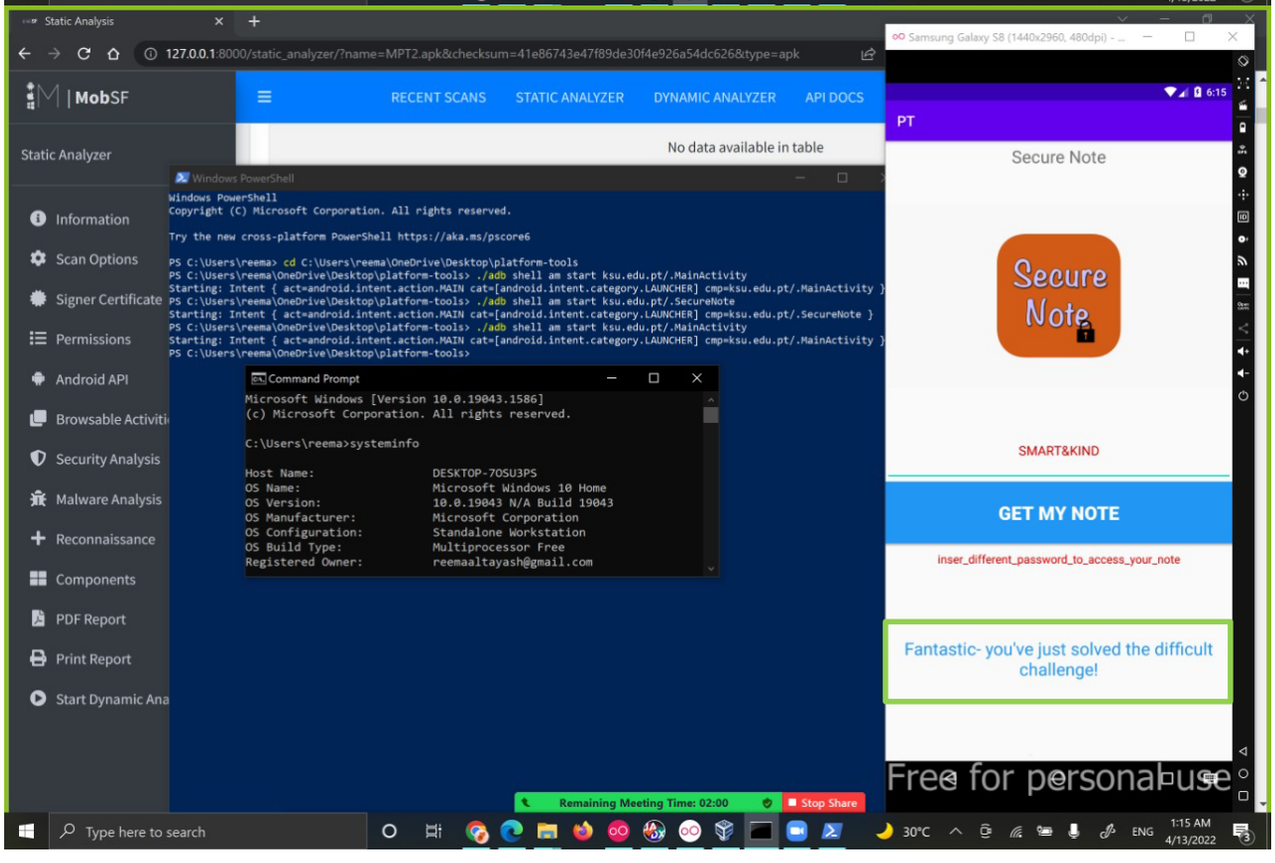
Recommendation	Access to the logs should only be granted to certain, highly privileged users, such as administrators. Also, Hash passwords using a good hashing algorithm.
Reference	**https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2 526Monitoring**

3.3.3 Improper authorization

Severity Level	**HIGH**				
Technical Impact	HIGH	**Technical Impact**		HIGH	
		Popularity	HIGH	**Simplicity**	HIGH
Observation	Using MobSF, we discovered a vulnerability in the MobSF's manifest analysis, the activity was unprotected and the export = true. This means that the activity was not exported properly (an improper rogue export is a child of an improper rogue authorization). Then we used the adb command to open the Secure Note page. SMART&KIND was displayed when the page was opened, and when we returned to the Main Activity page and entered the password SMART&KIND, this vulnerability was exploited successfully.				
Impact	This vulnerability is harmful because it leads to system corruption and unauthorized access to sensitive information. Improperly exporting activity is also risky, as malicious applications can exploit activity to gain access to sensitive information and perform malicious actions.				
Platform	**Android**				
Proof of Concept					







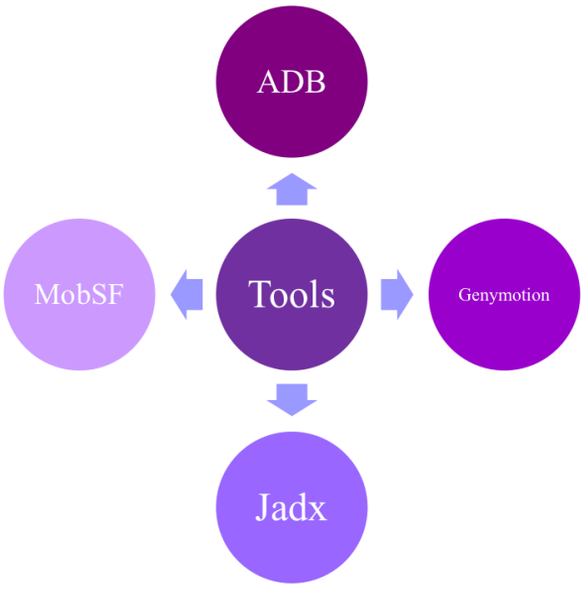
Recommendation	You should rely on permissions coming from the backend system rather than relying on the permissions coming from the mobile device. And also you should insure that you have restrictions on any exported operations.
Reference	**https://owasp.org/www-project-mobile-top-10/2016-risks/m6-insecure-authorization**

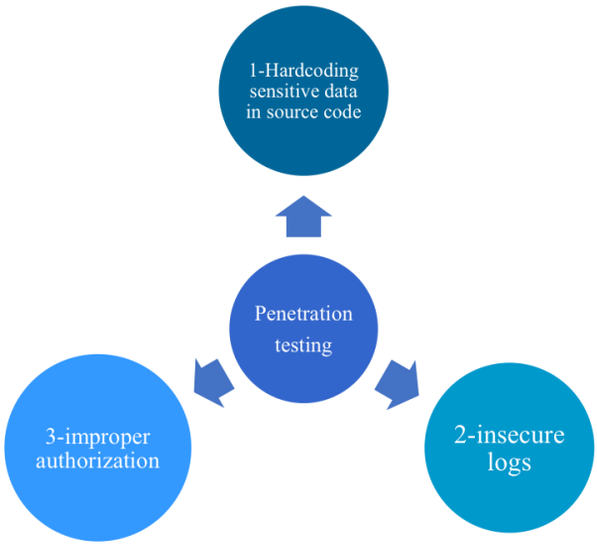
Appendix A: About the Team

Team Code: W1T6_7		
Student Name	**Serial Number**	Role: (NOTE: we worked on the project together via zoom)
Reema AlKraidees	24	vulnerability: Insecure logs, and common report parts.
Reema AlTayash	20	all work is done in her device and worked together on common report parts.
Reema AlOwerdi	4	vulnerability: improper authorization, and common report parts.
Layan AlZahrani	13	vulnerability: Hardcoding sensitive data in source code, and common report parts.

Appendix B: Our Methodology

Our methodology on Mobile penetration testing is based on Mobile Open Web Application Security Project (OWASP); Our assessment methodology to carry out the mobile penetration testing includes 2 phases as The first phase is setting the environment for penetration testing mobile applications as seen in the part below, and the second phase is implementing the penetration testing as explained previously in the report.





IT 371

PRE- MPT PROJECT PHASE

SUPERVISED BY: L. MANAL ALAQIL

PREPARED BY: W1TEAM7 AND W1TEAM6

REEMA ALKRAIDEES 24 REEMA ALTAYASH 20

REEMA ALOWERDI 4 LAYAN ALZAHRANI 13

SECTION: 52918

WHAT WE USED:

ANDROID STUDIO

provides code templates that follow the Android design and development best practices to get you on the right track to creating beautiful, functional apps.

GENYMOTION

is an Android emulator which includes a complete set of sensors and features in order to interact with a virtual Android environment.

JADX

is an unmaintained decompiler for the Java programming language.

MOBSF

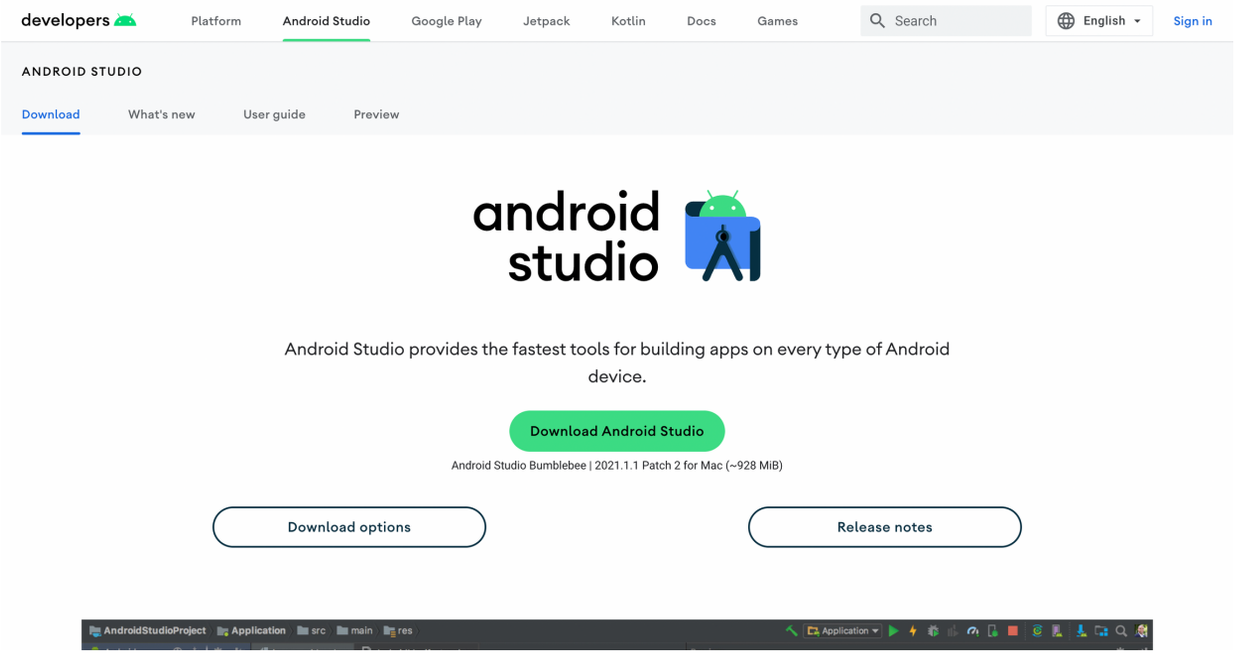
Mobile Security Framework is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

ADB

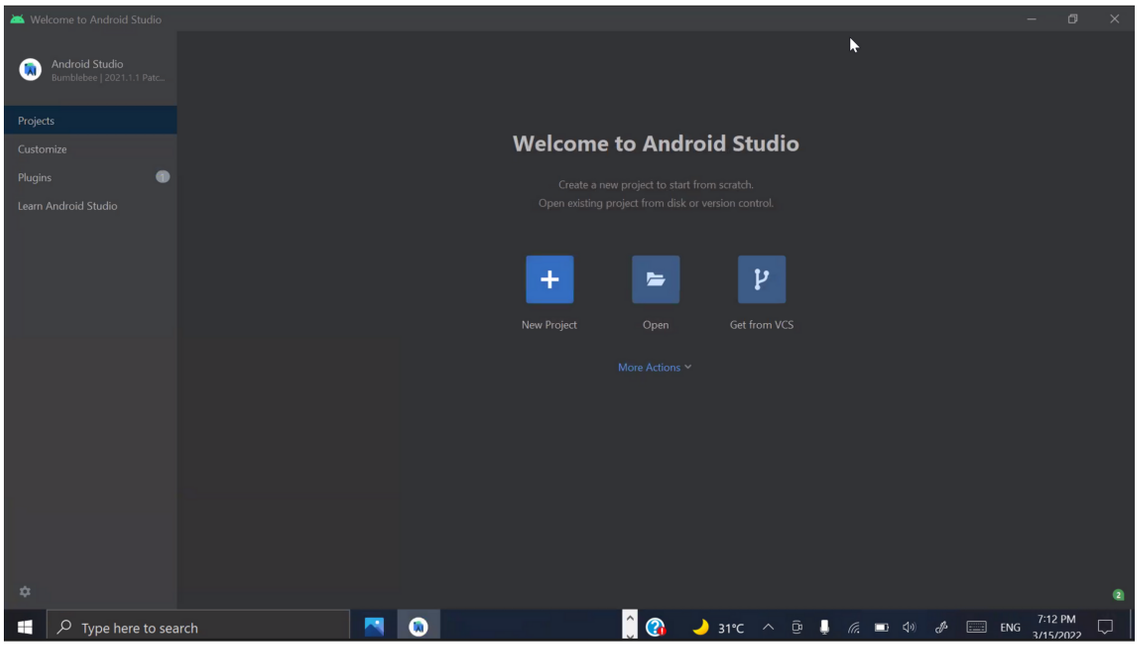
android debug bridge is a programming tool used for the debugging of Android-based devices.

Task#1: Get familiar (Review) Android code

1 INSTALL ANDROID STUDIO.

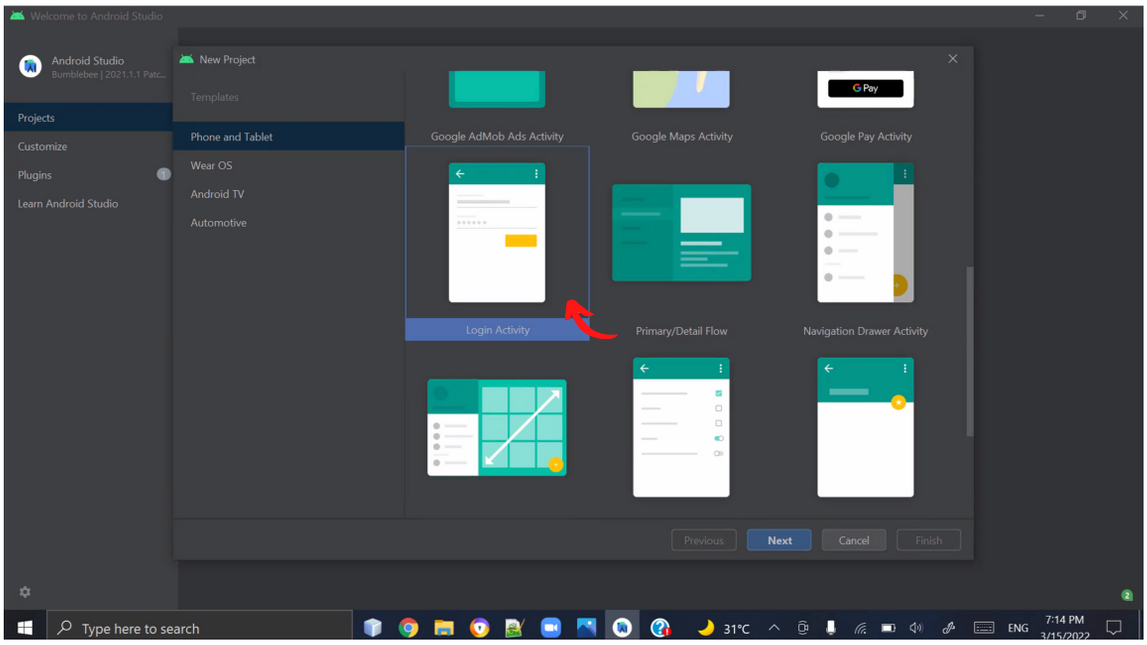


2 CREATE NEW PROJECT IN ANDROID STUDIO.

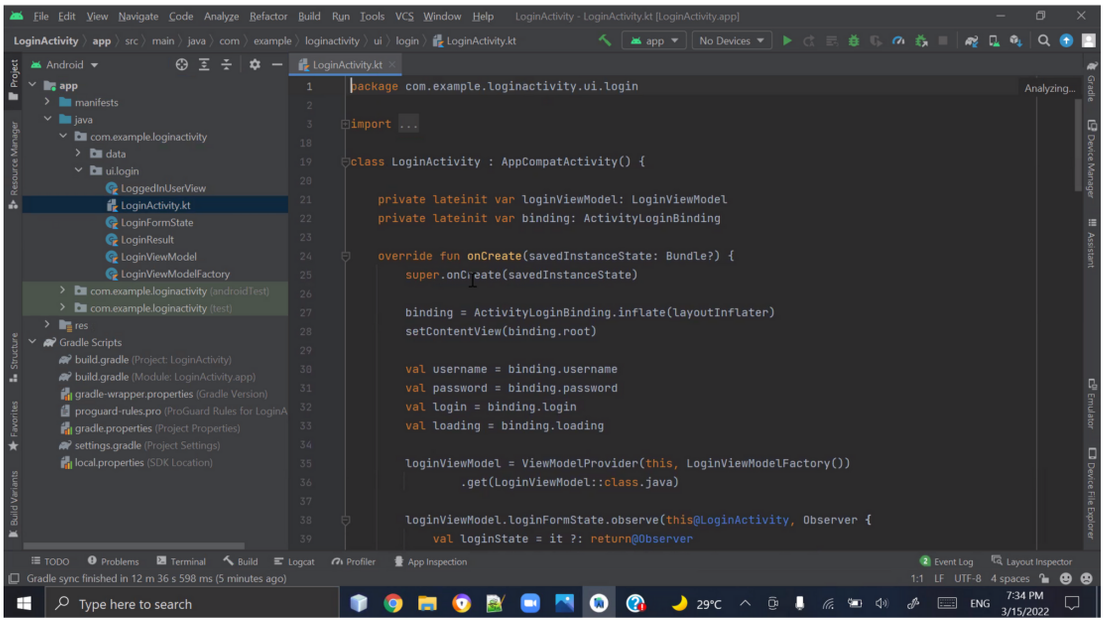


3 CHOOSE THE LOGIN ACTIVITY TEMPLATE (SUGGESTED)

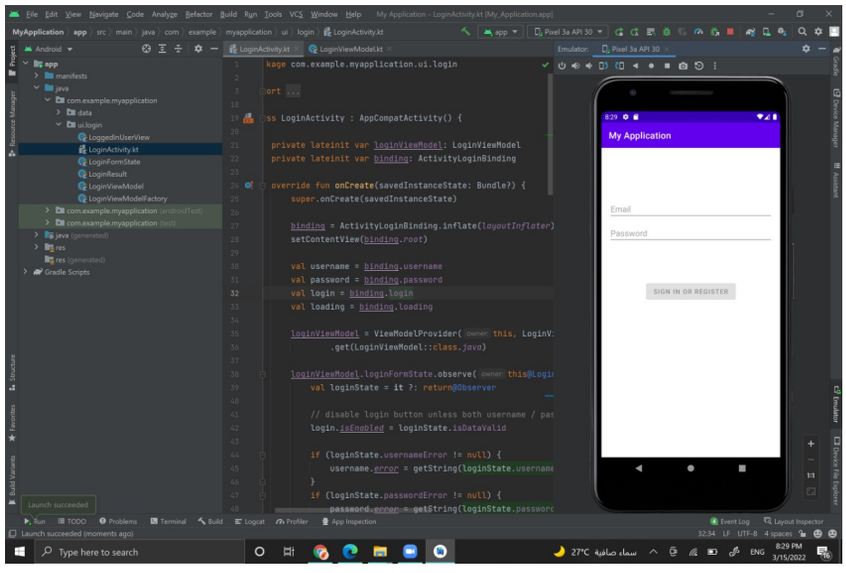
- (HAS A SIGN IN OR REGISTER BUTTON AND USER NAME AND PASSWORD FIELDS)



4 REVIEW THE CODE OF THE LOGIN ACTIVITY.



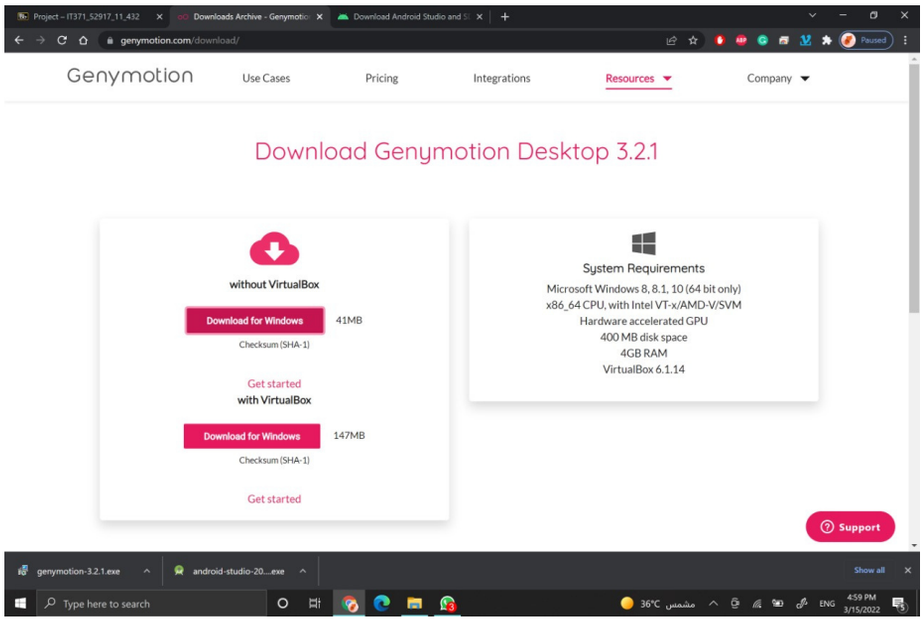
5 RUN THE SIMPLE ANDROID APP (LOGIN ACTIVITY)



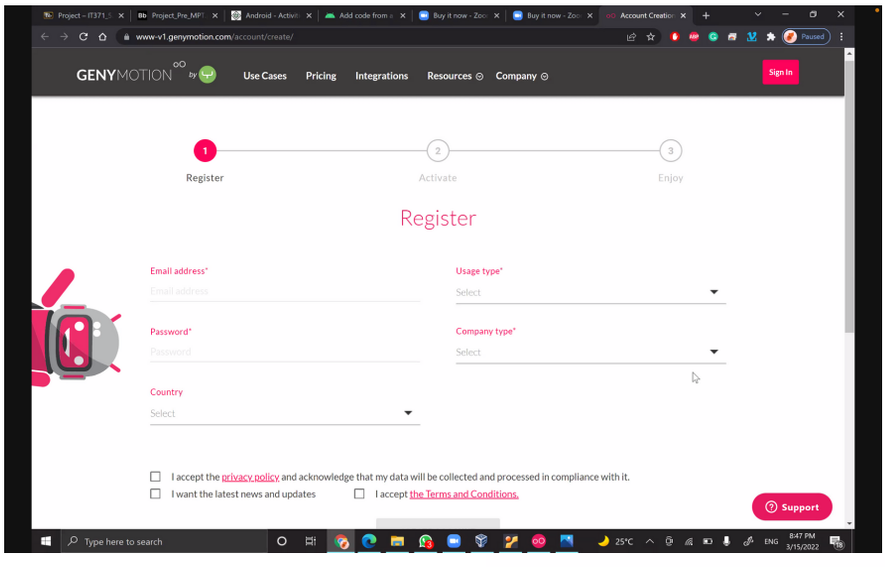
Task#2: Set up your MPT Environment:

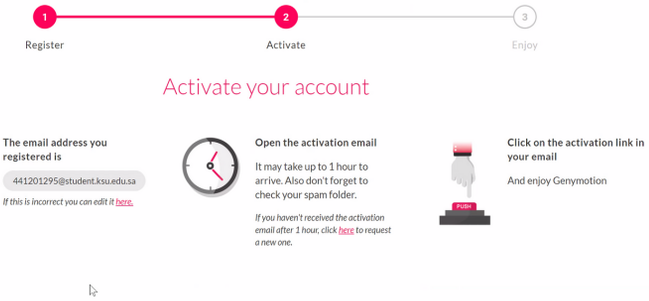
WE CREATED AN EMULATION OF THE ANDROID PHONE USING GENYMOTION.

1 INSTALL GENYMOTION (WITHOUT VIRTUAL BOX)



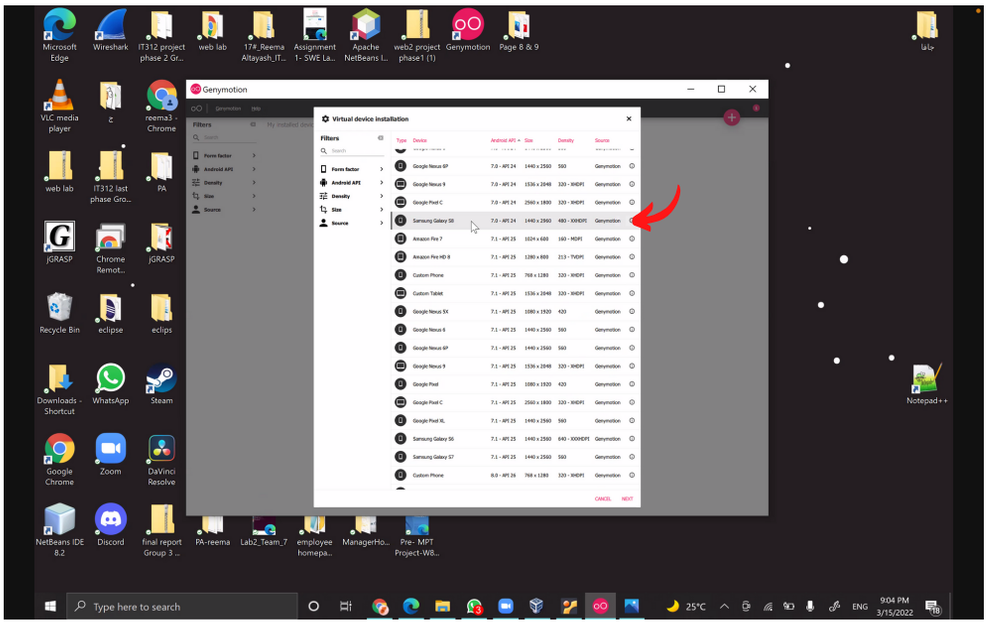
2 CREATE A NEW ACCOUNT IN GENYMOTION



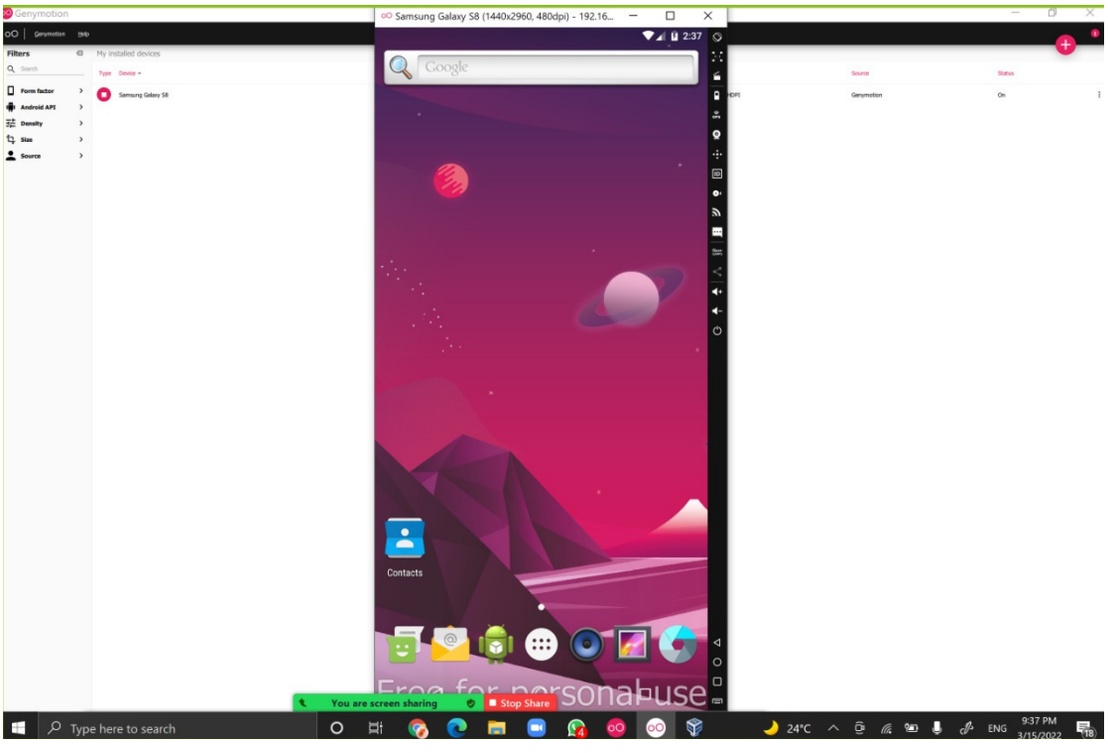


3 SELECT THE + BUTTON THEN CHOOSE A VIRTUAL DEVICE

- WE CHOSE THE SAMSUNG GALAXY S8

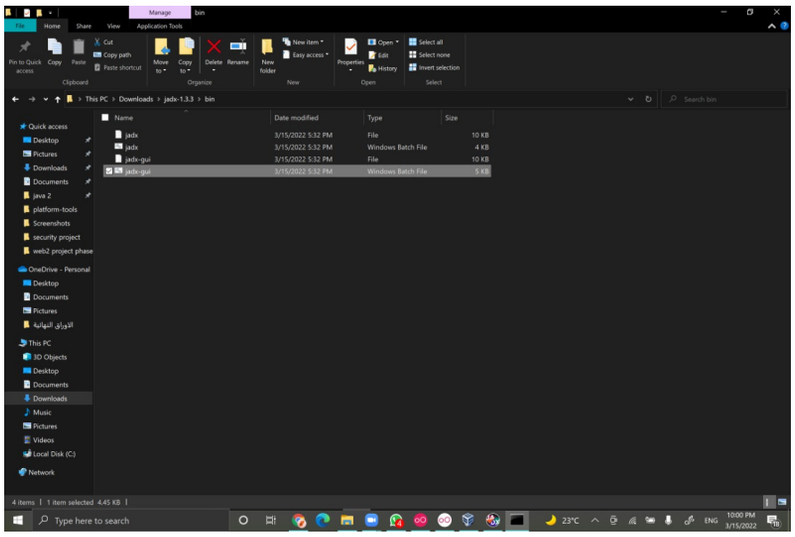


4 SET UP THE EMULATION DEVICE

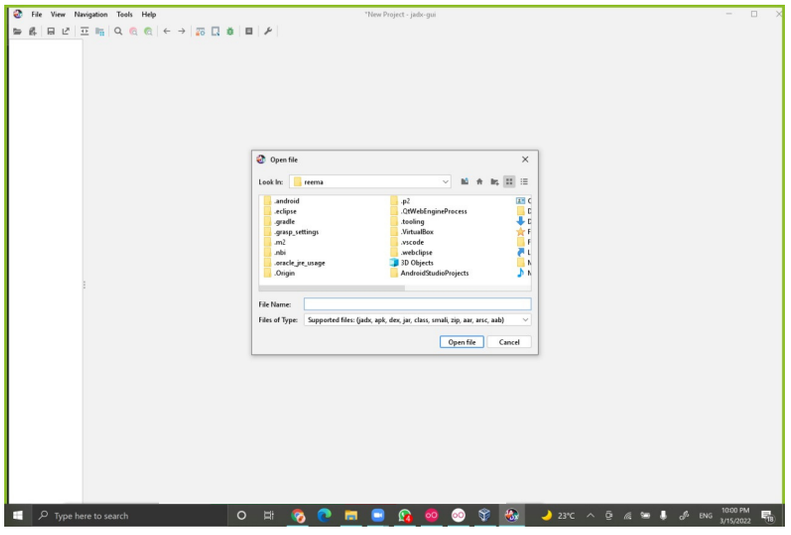


JDK SETUP

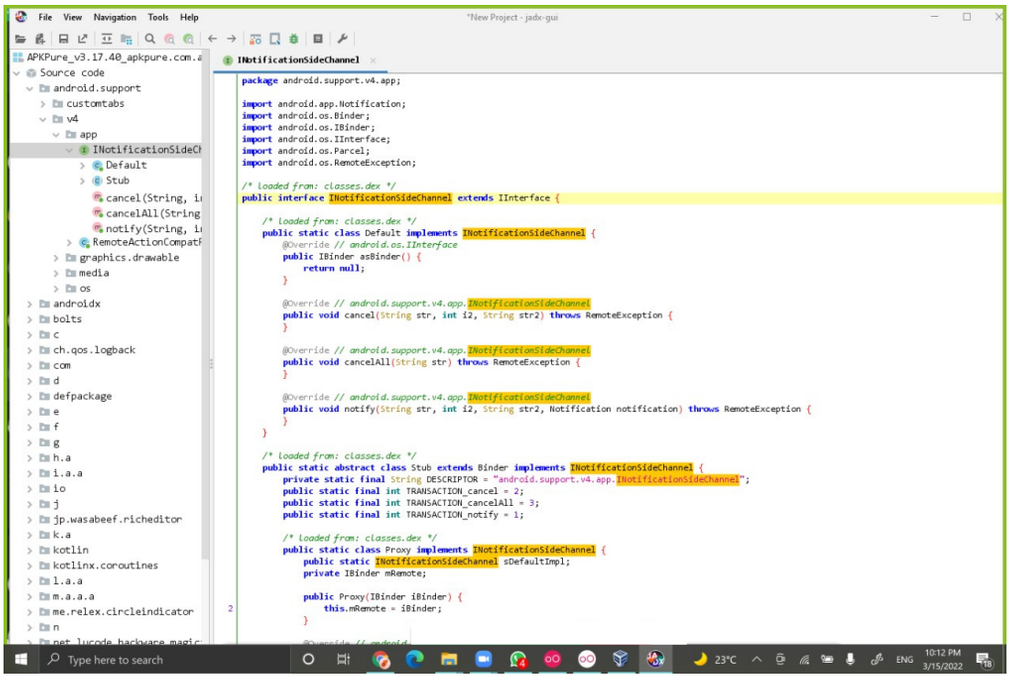
1 WE DOWNLOADED THE ZIP FILE, OPENED IT , AND CLICKED ON JADX.GUI



2 WE SELECTED THE APK FILE TO DECOMPILE AND READ SOURCE CODE



3 SOURCE CODE (TO READ)



MobSF SETUP

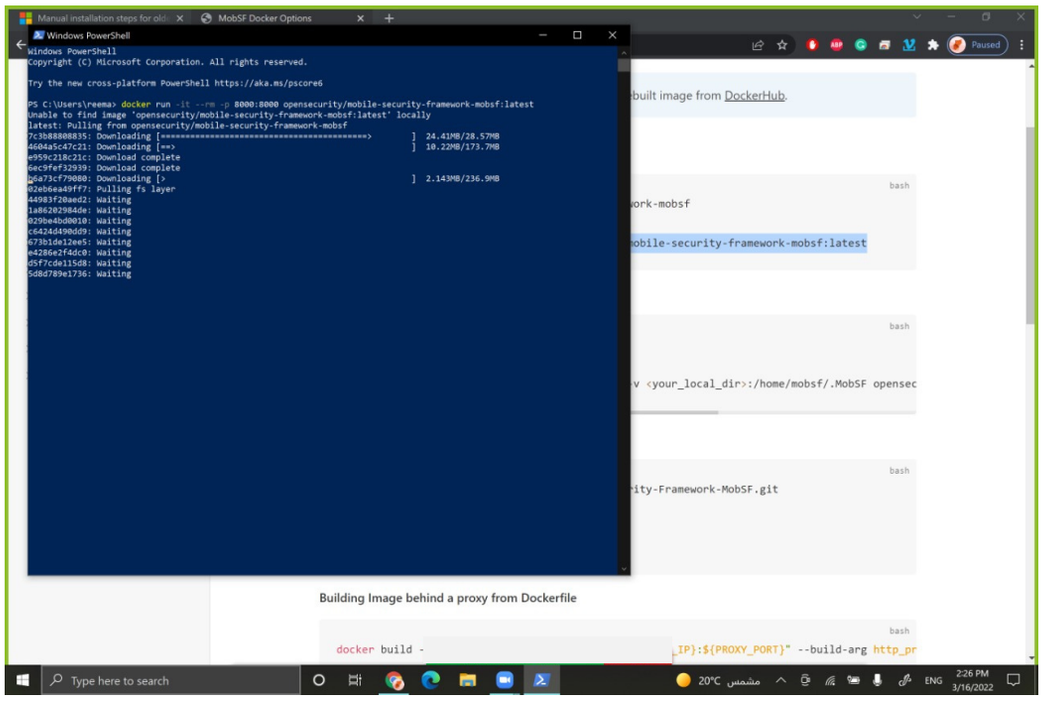
PREREQUISITES TO SETTING UP MOBSF

- INSTALL DOCKER ENGINE

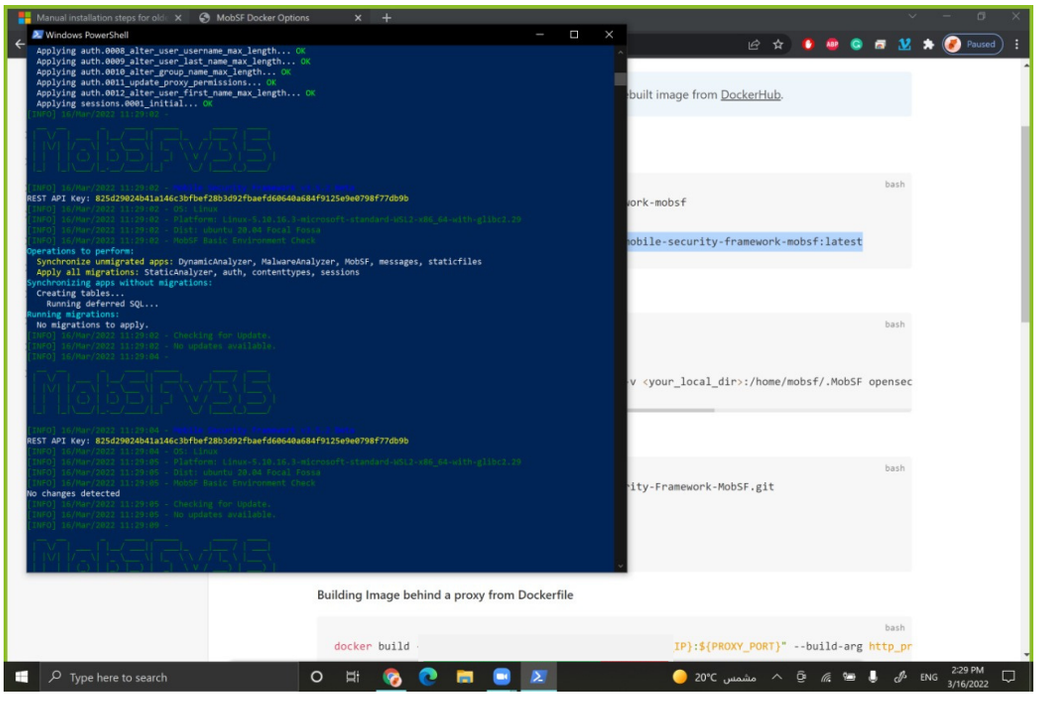
1 OPEN MICROSOFT POWERSHELL TO WRITE

ENTER THE COMMAND

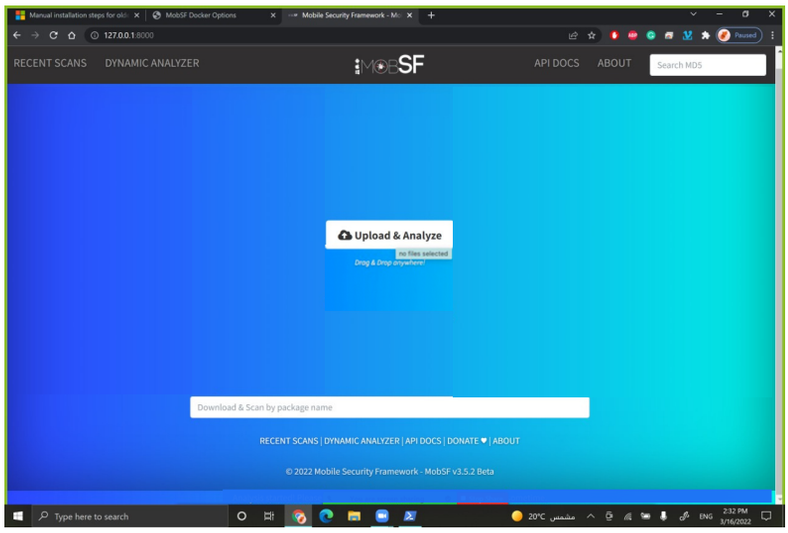
DOCKER RUN -IT --RM -P 8000:8000 OPENSECURITY/MOBILE-SECURITY-FRAMEWORK-MOBSF:LATEST



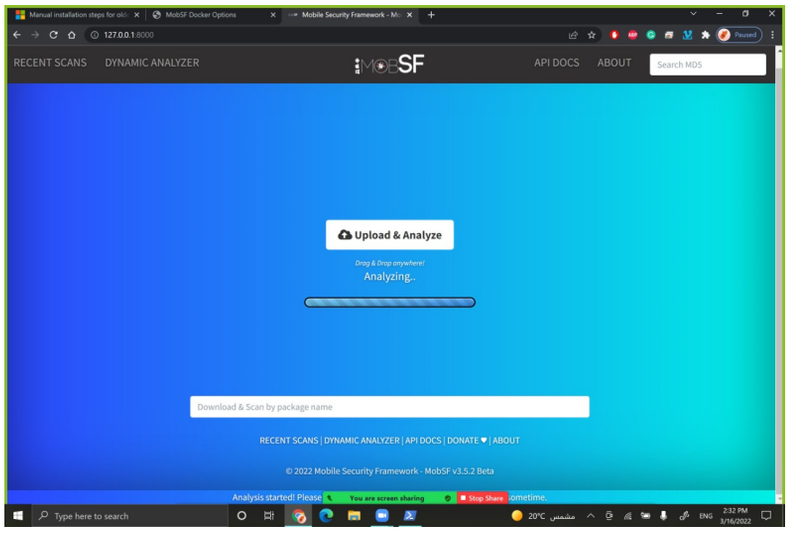
2 MOBSF RUNNING

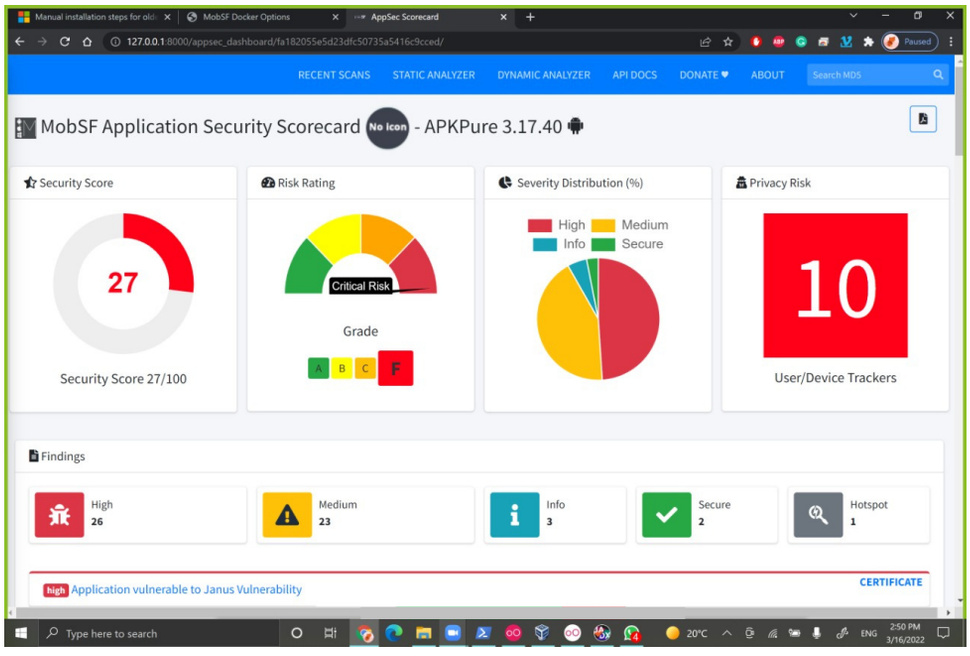


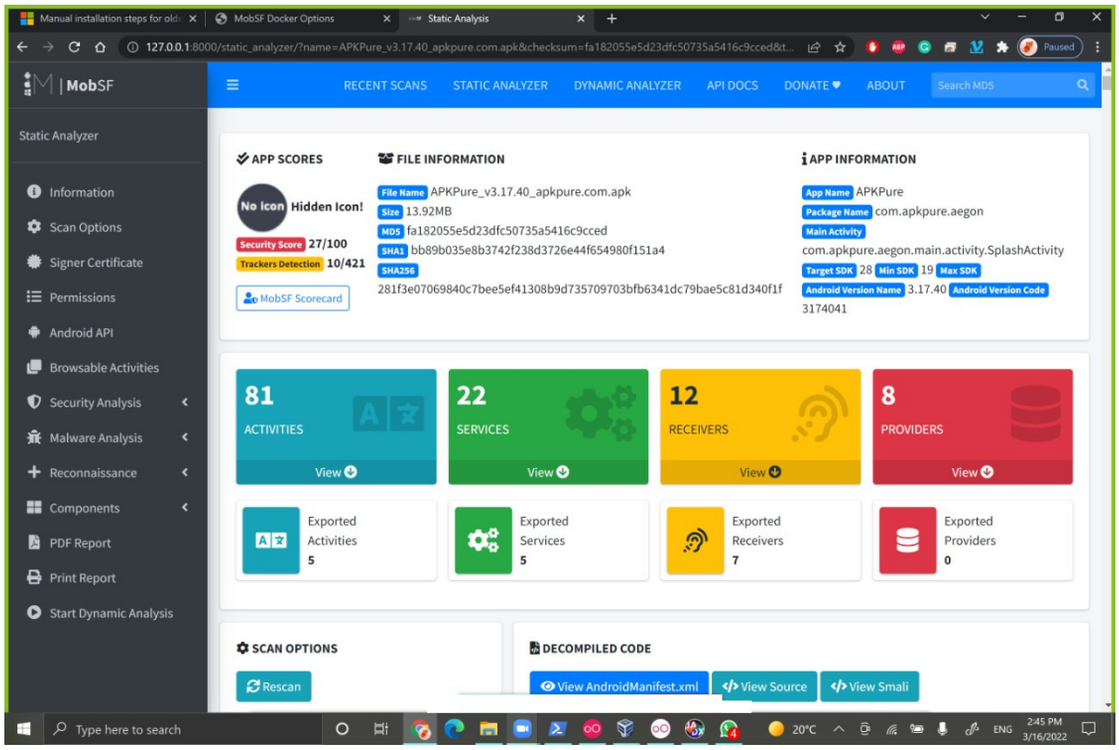
3 BY RUNNING THE MOBSF WE OPENED THE WEB PAGE

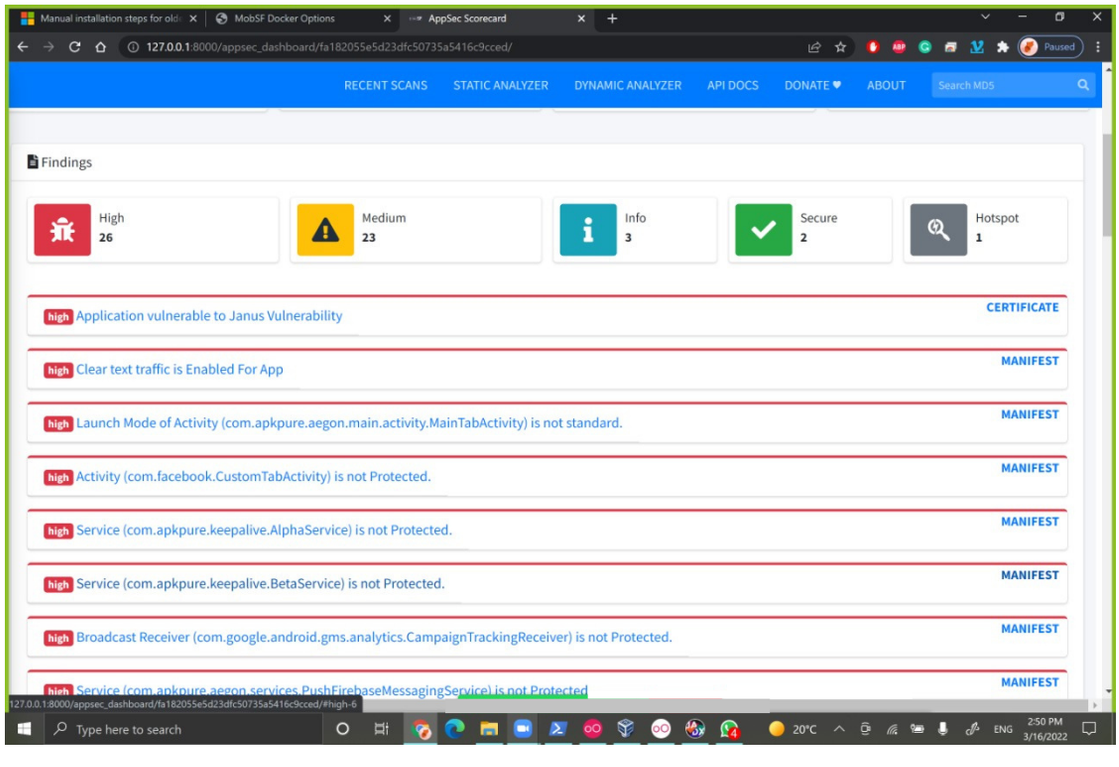


4 TO ANALYZE THE .APK FILE WE PRESSED (UPLOAD AND ANALYZE) TO AUTOMATICALLY FIND VULNERABILITIES



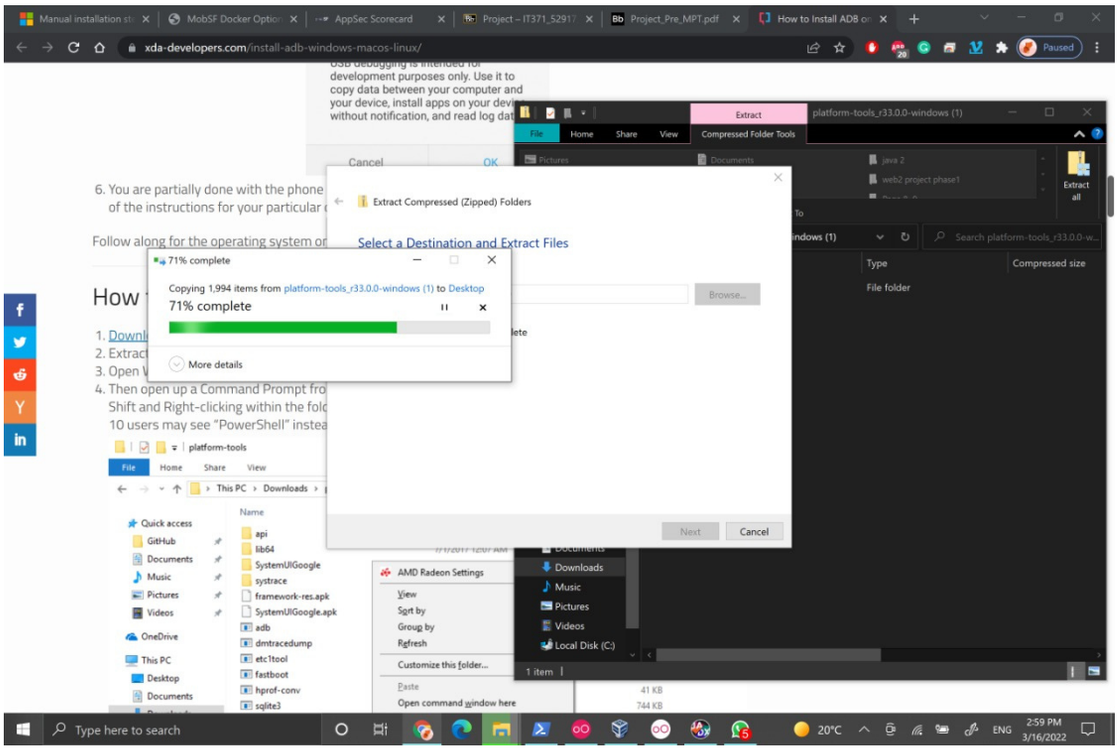



MobSF
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API DOCS
DONATE
ABOUT
Search MD5
APP SCORES
No Icon
Hidden Icon!
Security Score 27/100
Trackers Detection 10/421
MobSF Scorecard
FILE INFORMATION
File Name APKPure_v3.17.40_apkpure.com.apk
Size 13.92MB
MD5 fa182055e5d23dfc50735a5416c9cced
SHA1 bb89b035e8b3742f238d3726e44f654980f151a4
SHA256
281f3e07069840c7bee5ef41308b9d735709703bfb6341dc79bae5c81d340f1f
APP INFORMATION
App Name APKPure
Package Name com.apkpure.aegon
Main Activity
com.apkpure.aegon.main.activity.SplashActivity
Target SDK 28 Min SDK 19 Max SDK
Android Version Name 3.17.40 Android Version Code
3174041
81
ACTIVITIES
View
22
SERVICES
View
12
RECEIVERS
View
8
PROVIDERS
View
Exported Activities 5
Exported Services 5
Exported Receivers 7
Exported Providers 0
SCAN OPTIONS
Rescan
DECOMPILED CODE
View AndroidManifest.xml
View Source
View Smali


RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API DOCS
DONATE
ABOUT
Search MD5
Findings
High 26
Medium 23
Info 3
Secure 2
Hotspot 1
high Application vulnerable to Janus Vulnerability CERTIFICATE
high Clear text traffic is Enabled For App MANIFEST
high Launch Mode of Activity (com.apkpure.aegon.main.activity.MainTabActivity) is not standard. MANIFEST
high Activity (com.facebook.CustomTabActivity) is not Protected. MANIFEST
high Service (com.apkpure.keepalive.AlphaService) is not Protected. MANIFEST
high Service (com.apkpure.keepalive.BetaService) is not Protected. MANIFEST
high Broadcast Receiver (com.google.android.gms.analytics.CampaignTrackingReceiver) is not Protected. MANIFEST
high Service (com.apkpure.aegon.services.PushFirebaseMessagingService) is not Protected MANIFEST

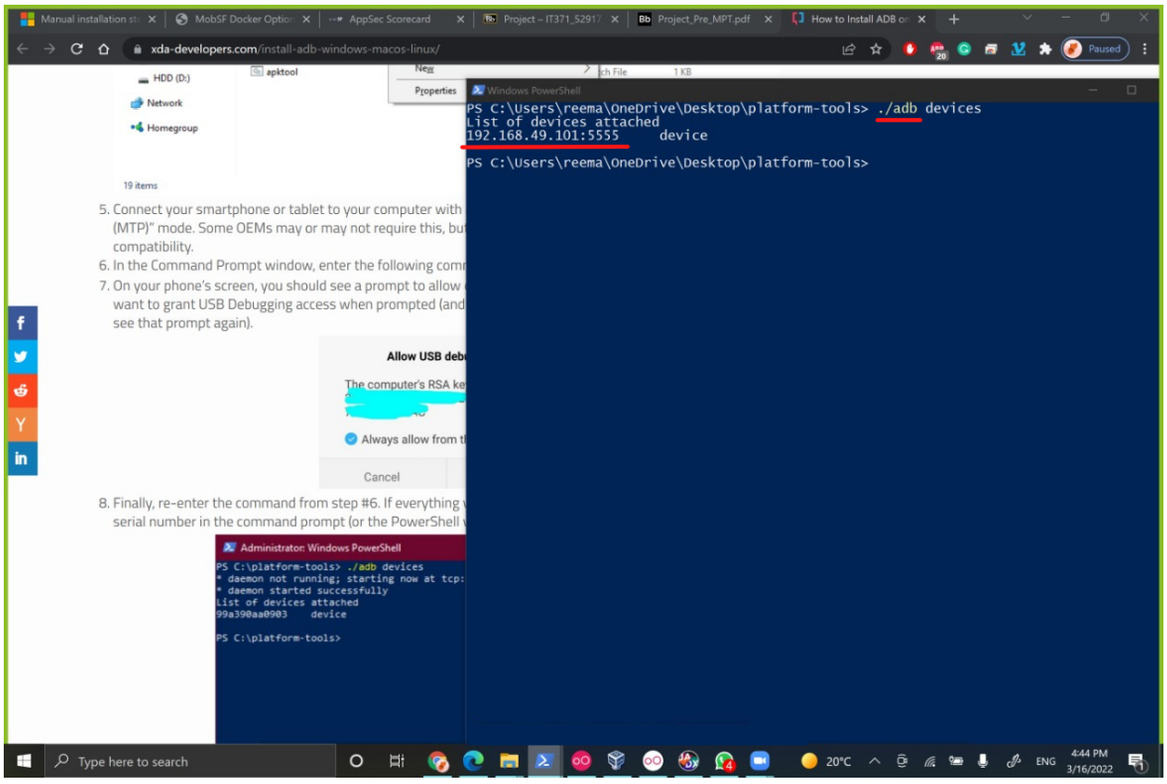
ADB SETUP

1 INSTALL ANDROID SDK PLATFORM TOOLS



2 WHILE ANDROID DEVICE IS RUNNING THROUGH GENYMOTION WE RAN THE MICROSOFT POWERSHELL AND WROTE ./ADB DEVICES TO INDICATE THE DEVICE IS COMMUNICATING THROUGH COMMAND LINE AND PLATFORM

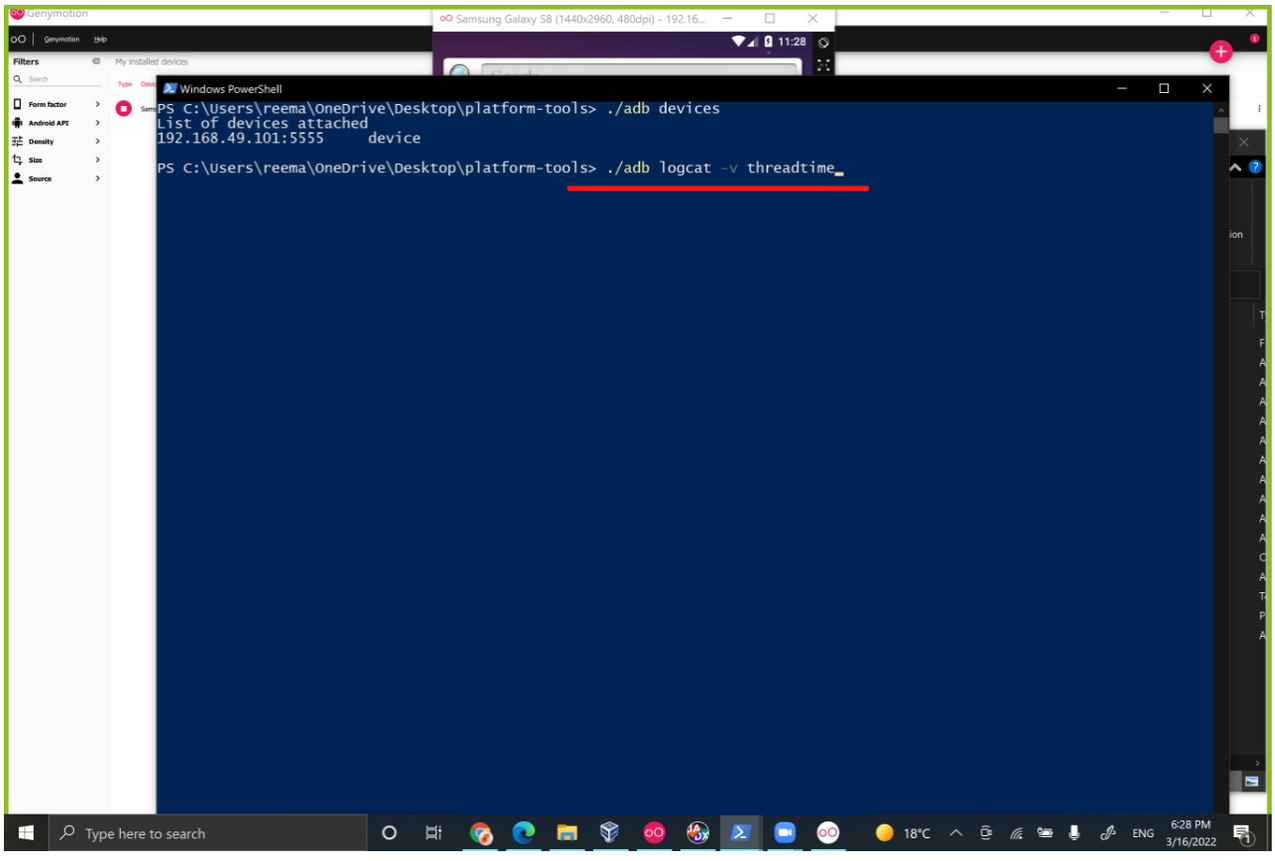
- THE SERIAL NUMBER 192.168.49.101:5555 IS SHOWN; INDICATING SUCCESSFUL COMMUNICATION



3 AFTER SUCCESSFUL CONNECTION WAS REACHED WE PREVIEWED COMMUNICATION BY TRYING TO SHOW THE LOGS OF THE ANDROID DEVICE

- (SAMSUNG GALAXY S8)

USING THE COMMAND "ADB LOGCAT -V THREADTIME"



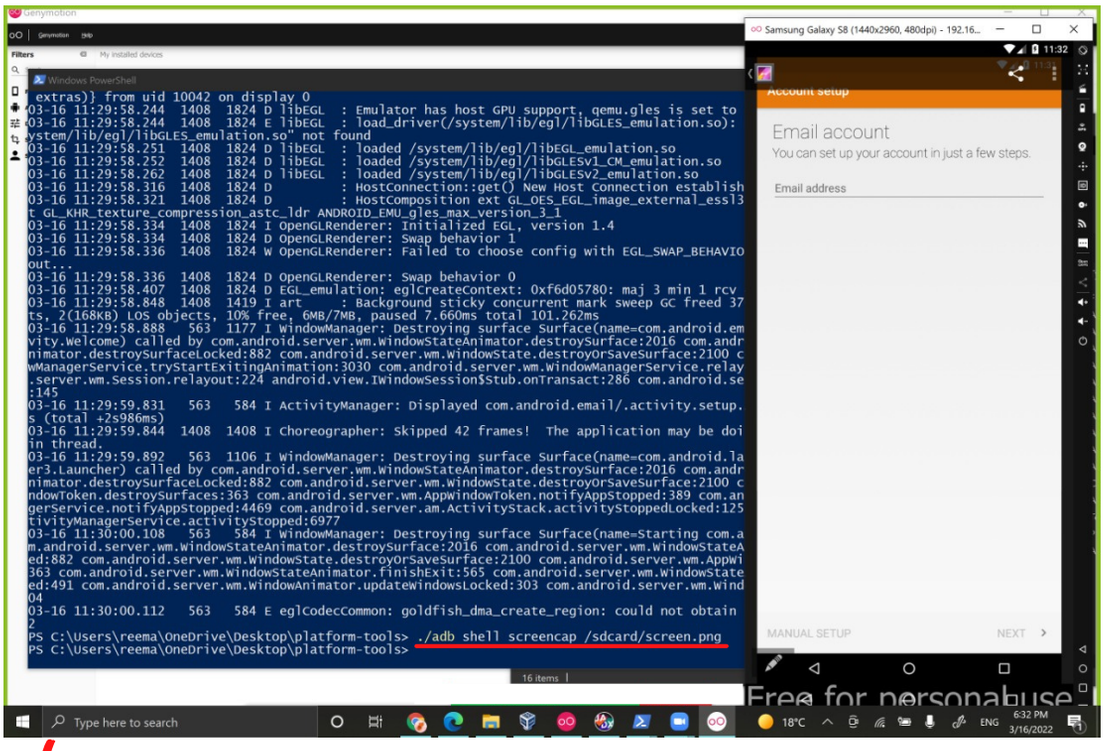
- *"LOGCAT" USED TO VIEW LOG OUTPUT*
- *"-V" USED TO SPECIFY A FORMAT MODIFIER*
- *"THREADTIME" USED TO DISPLAY THE DATE, INVOCATION TIME, PRIORITY, TAG, PID AND TID OF THREAD ISSUING THE MESSAGE.*

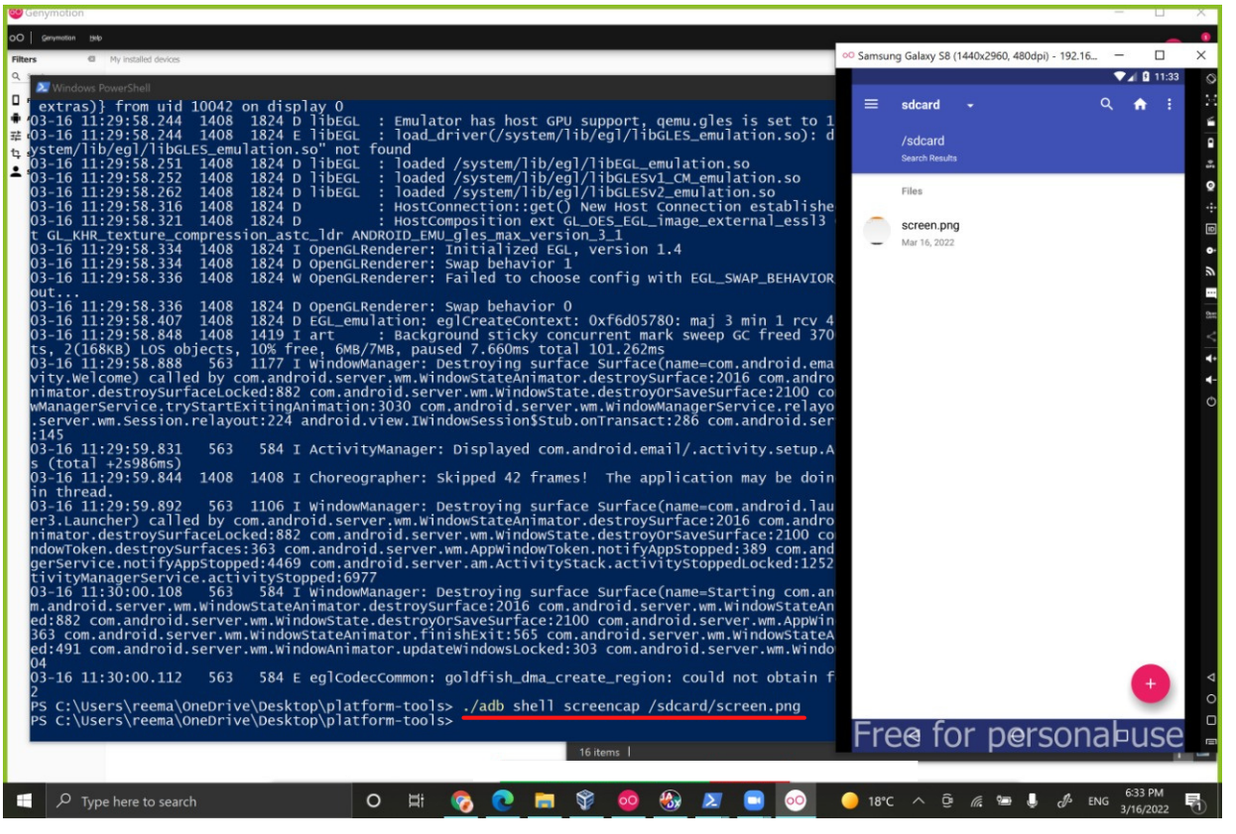
AFTER EXECUTING THE COMMAND , ANY CHANGES DONE ON THE EMULATOR WERE SHOWN IN THE LOGS.

4 TO TEST OUR WORK WE ATTEMPTED TO LAUNCH AN ACTION/ACTIVITY BY WRITING A COMMAND IN THE POWERSHELL TO MAKE SURE IT IS EXECUTED IN THE ANDROID DEVICE EMULATOR AS WELL

WE CHOSE TO DO A SCREENSHOT OF THE DEVICE AND SAVE THE PICTURE IN A SPECIFIC FILE IN THE DEVICE USING THE COMMAND LINE:

"ADB SHELL SCREENCAP /SDCARD/SCREEN.PNG"





TECHNICAL ISSUES WE FACED IN PHASE 1:

WE INSTALLED ANDROID STUDIO EASILY WITHOUT ANY DIFFICULTIES.

AS FOR GENYMOTION, WE COULDN'T GET OUR EMULATOR DEVICE (SAMSUNG GALAXY S8) TO WORK AT FIRST, WE KEPT GETTING ERROR MESSAGES EVERY TIME WE TRY AND START THE DEVICE, WE HAD TO REBOOT THE LAPTOP, THEN WE DECIDED TO DECLINE THE PERMISSIONS GENYMOTION ASKED FOR UNLIKE OUR FIRST FEW TRIES AND IT FINALLY WORKED.

THE JADX DOWNLOAD AND EXECUTION RAN SMOOTHLY.

WITH MOBSF WE FIRST TRIED TO SET IT UP BY FOLLOWING THE INSTRUCTIONS IN THE GIVEN RESOURCE, WE DOWNLOADED ALL THE PREREQUISITES NEEDED FOR SETTING UP MOBSF BUT UNFORTUNATELY IT DIDN'T WORK. WE SEARCHED FOR AN ALTERNATIVE ON HOW TO SETUP MOBSF, AND WE FOUND THAT WE COULD DO IT BY INSTALLING DOCKER FOR DESKTOP, AFTER THAT THE SETUP OF MOBSF WAS EASY AND IT EXECUTED WELL.

FOR ADB WE HAD TO RESTART THE COMPUTER, CAUSING US TO HAVE TO RESTORE ALL OUR WORK UP TO THE LAST CHECKPOINT, BUT OTHER THAN THAT EVERYTHING ELSE WENT FINE.

REFERENCES:

- HTTPS://EN.WIKIPEDIA.ORG/WIKI/ANDROID_DEBUG_BRIDGEIN STALL PYTHON 3.8-3.9
- HTTPS://MOBSF.GITHUB.IO/DOCS/#/
- HTTPS://EN.WIKIPEDIA.ORG/WIKI/JAD_(SOFTWARE)
- HTTPS://DOCS.GENYMOTION.COM/DESKTOP/
- HTTPS://DEVELOPER.ANDROID.COM/STUDIO/PROJECTS/TEMPL ATES#LOGINACTIVITY
- HTTPS://WWW.TUTORIALSPOINT.COM/ANDROID/ANDROID_ACI TIVITIES.HTM
- HTTPS://WWW.GENYMOTION.COM/FUN-ZONE/
- HTTPS://GITHUB.COM/SKYLOT/JADX
- HTTPS://GITHUB.COM/MOBSF/MOBILE-SECURITY- FRAMEWORK-MOBSF
- HTTPS://YOUTU.BE/LPXVZINCJE8
- HTTPS://WWW.XDA-DEVELOPERS.COM/INSTALL-ADB- WINDOWS-MACOS-LINUX/

WORK DISTRIBUTION:

REEMA ALKRAIDEES [441203734] --> REPORT + TASK 1 + GENYMOTION HELP WITH EXECUTION

REEMA ALTAYASH [441201295] --> TASK 2 ALL (JDK , MOBSF , ADB)

LAYAN ALZAHRANI [441200974] --> TASK 2 ALL (JDK , MOBSF, ADB)

REEMA ALOWERDI [439200449] --> REPORT + MOBSF HELP WITH EXECUTION