Malicious APK File Analysis

No. 15

Stage 2

Your job is to investigate the content of a given malicious Android app. Using the tools given in the above and below link and the ones presented in the class to learn about analysis of apk file, your job is to analyze the given app and reveal the secret code along with the description of what the malicious part is trying to do:

https://github.com/asiamina/A-Course-on-DigitalForensics/tree/master/Hands%20On%20Experience/Android%20Forensics

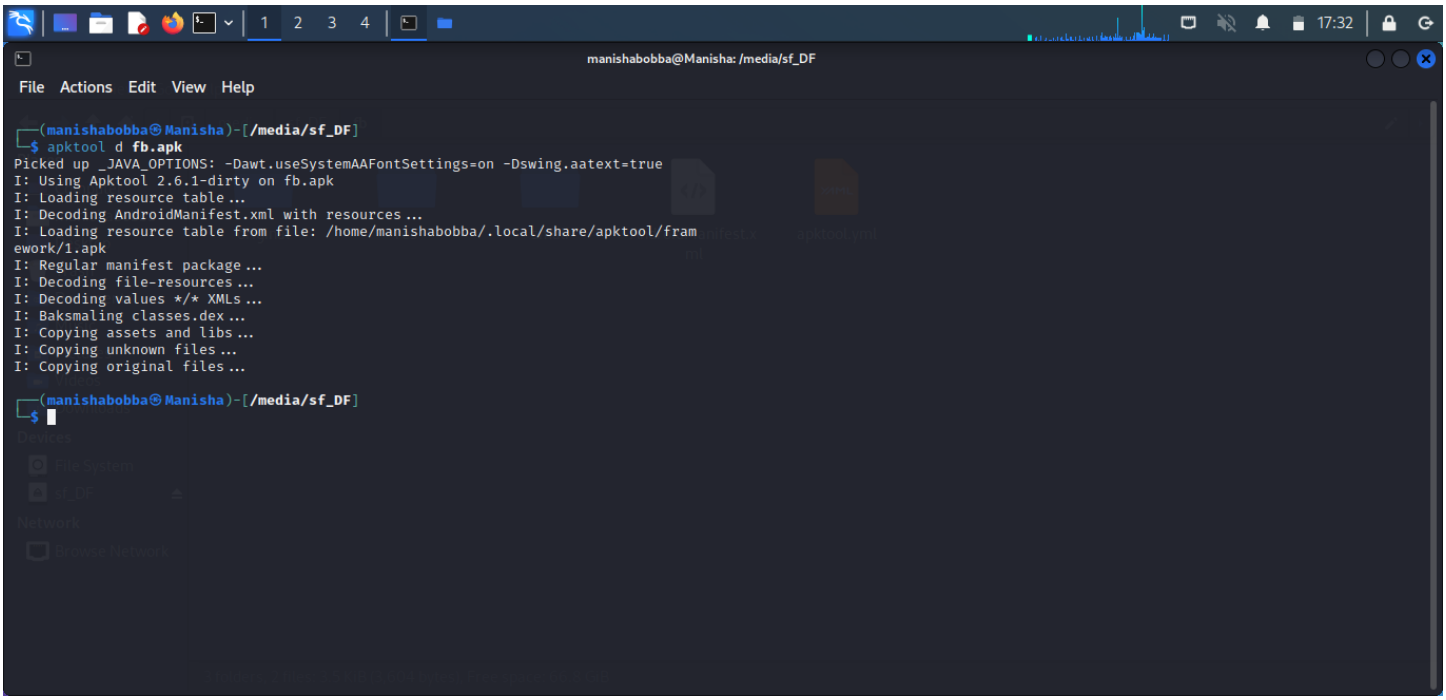
1. Analyze the given app and report what the malicious part is trying to achieve. Submit a word document along with some snapshots.

2. What is the secret code? Submit a report step-by-step explaining on how you have done the process and which tools are used and how.

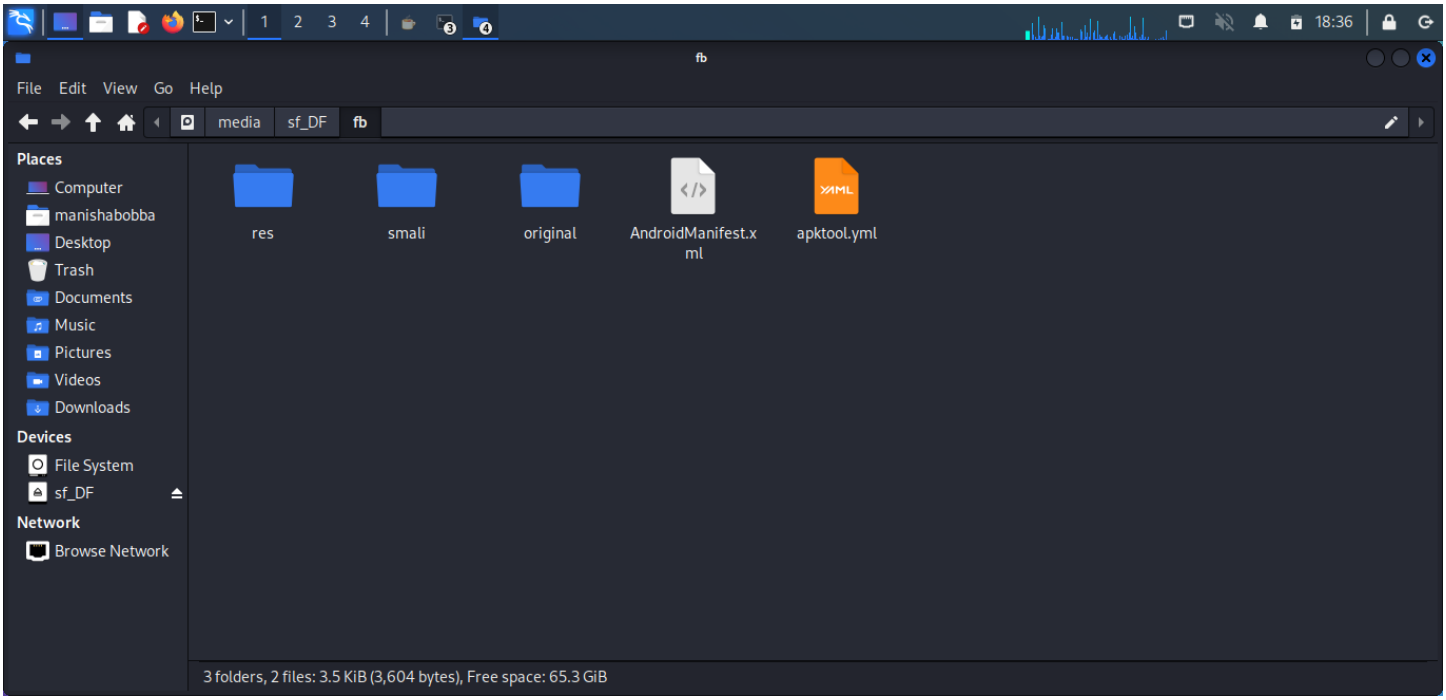
Deliverable: A document showing your analysis of the android app file along with some snapshots and steps taken and revealing secret code in the end.

Team given to analyze – Team 15

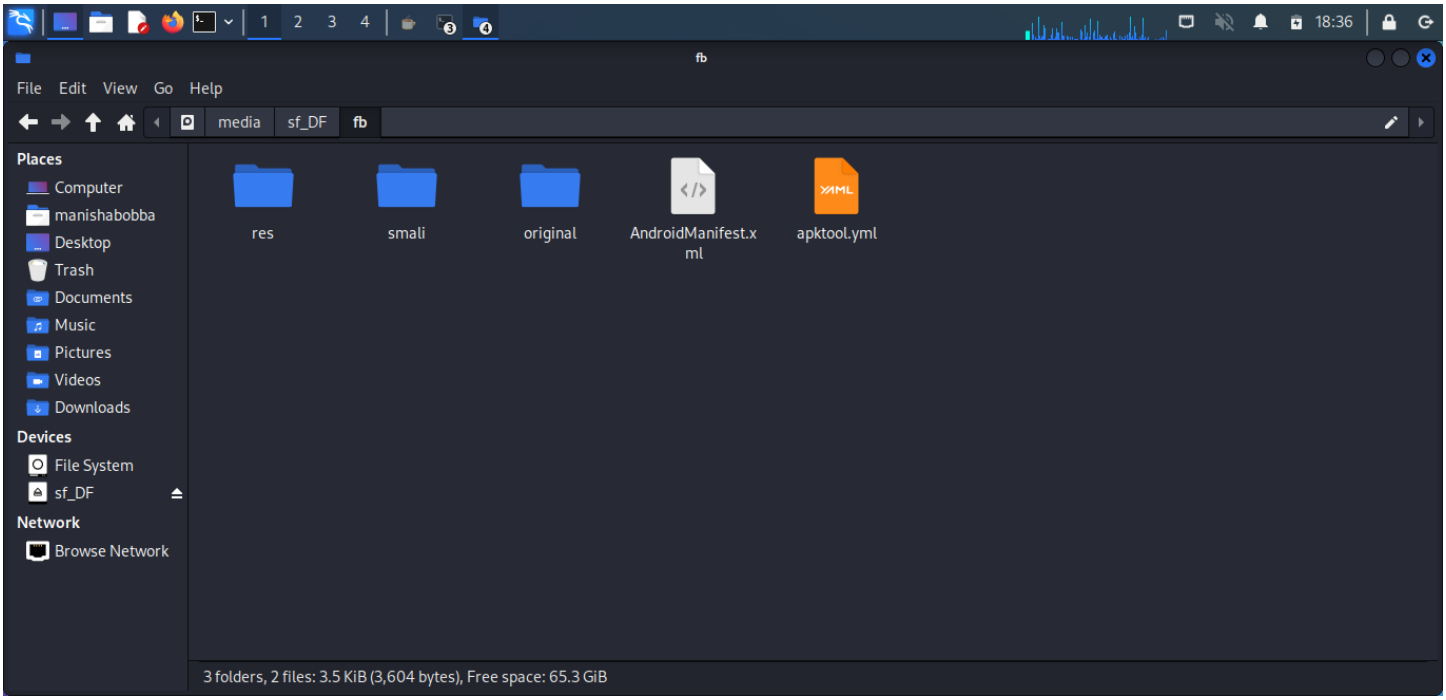
We used APK tool to decompile the APK.



After decompiling the APK file we can see the folders below.

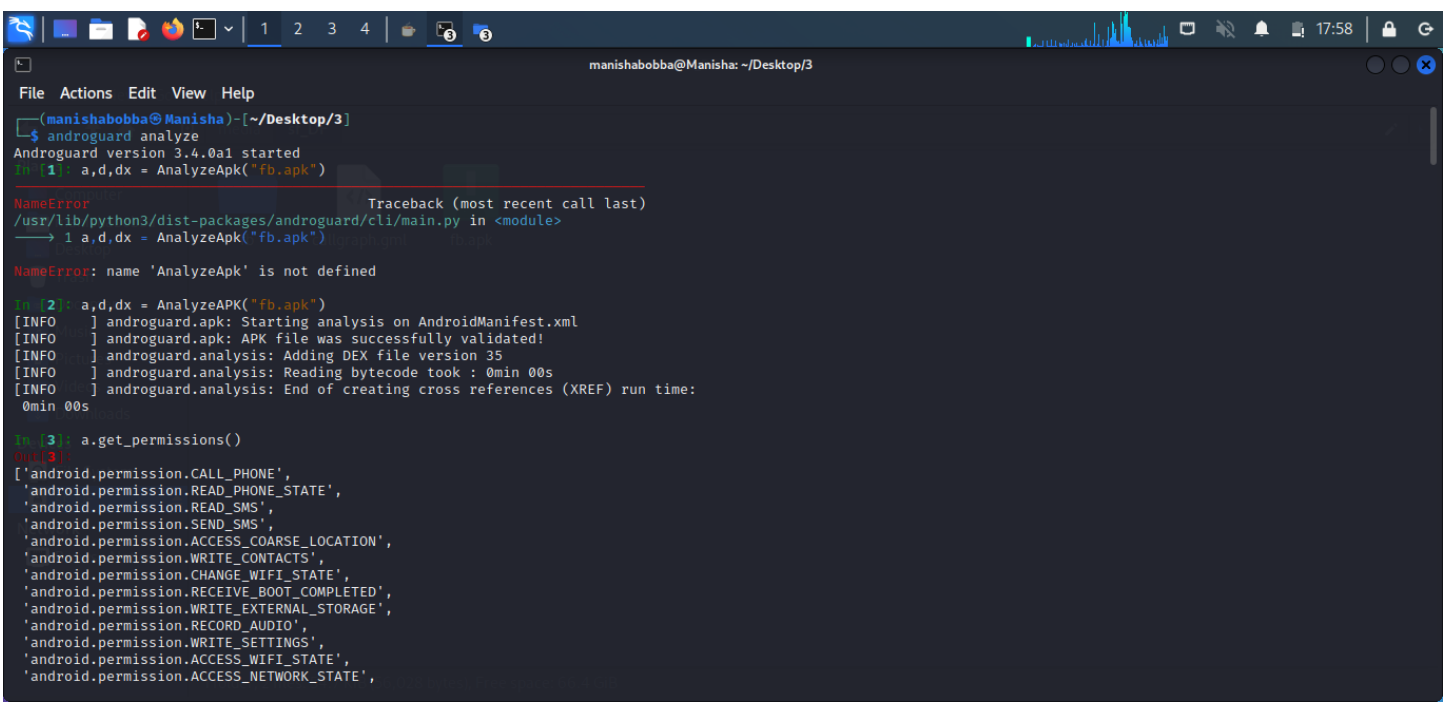


We searched the secret code everywhere in the file but couldn't find it. It looks like Team 15 has not embedded any secret code in the file.



Analysis of the APK File

We used androguard to do the static analysis of the APK file.



These are the permissions that app needs to get, in order to use.

We used ***a.get_permissions()*** method to get all permissions that the APK uses.

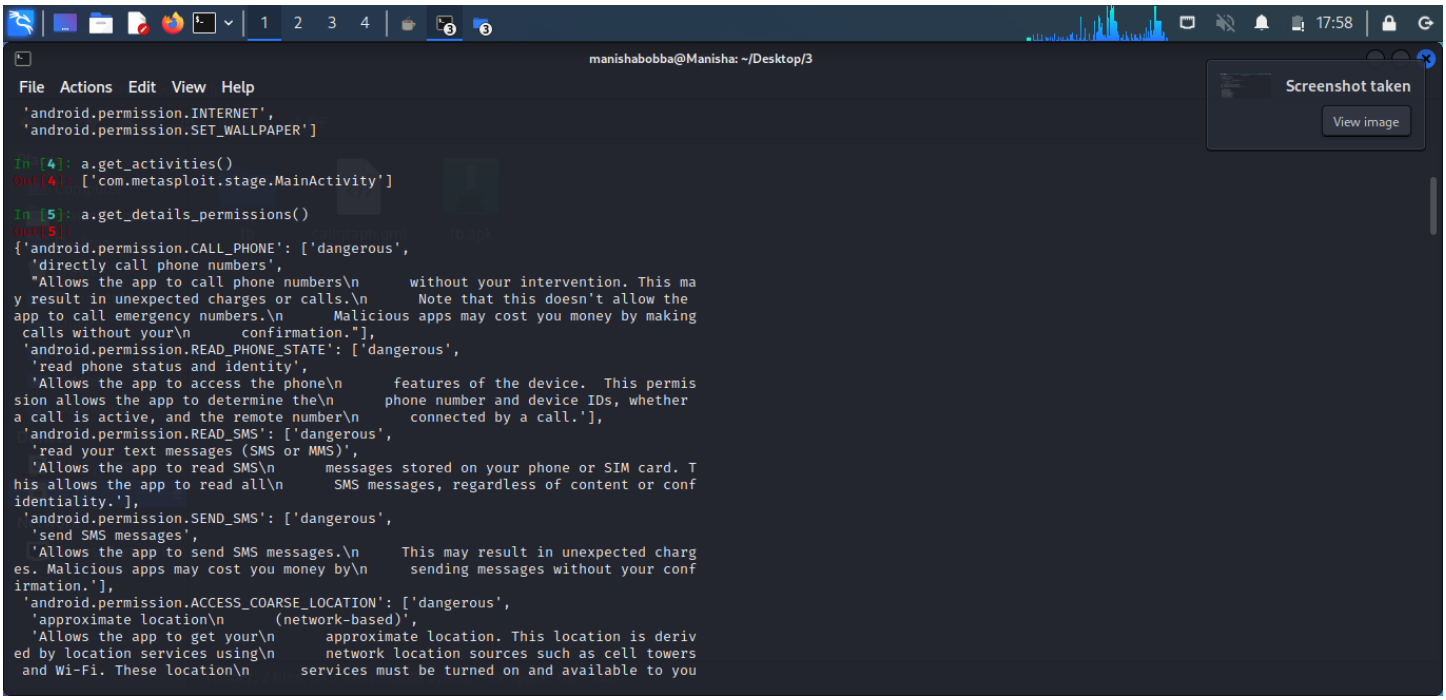
```
In [3]: a.get_permissions()
Out[3]:
['android.permission.CALL_PHONE',
 'android.permission.READ_PHONE_STATE',
 'android.permission.READ_SMS',
 'android.permission.SEND_SMS',
 'android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.WRITE_CONTACTS',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.WRITE_EXTERNAL_STORAGE',
 'android.permission.RECORD_AUDIO',
 'android.permission.WRITE_SETTINGS',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS',
 'android.permission.WRITE_CALL_LOG',
 'android.permission.WAKE_LOCK',
 'android.permission.READ_CONTACTS',
 'android.permission.CAMERA',
 'android.permission.ACCESS_FINE_LOCATION',
 'android.permission.READ_CALL_LOG',
 'android.permission.RECEIVE_SMS',
 'android.permission.INTERNET',
 'android.permission.SET_WALLPAPER']

In [4]: a.get_activities()
Out[4]: ['com.metasploit.stage.MainActivity']

In [5]: a.get_details_permissions()
Out[5]:
{'android.permission.CALL_PHONE': ['dangerous',
  'directly call phone numbers',
```

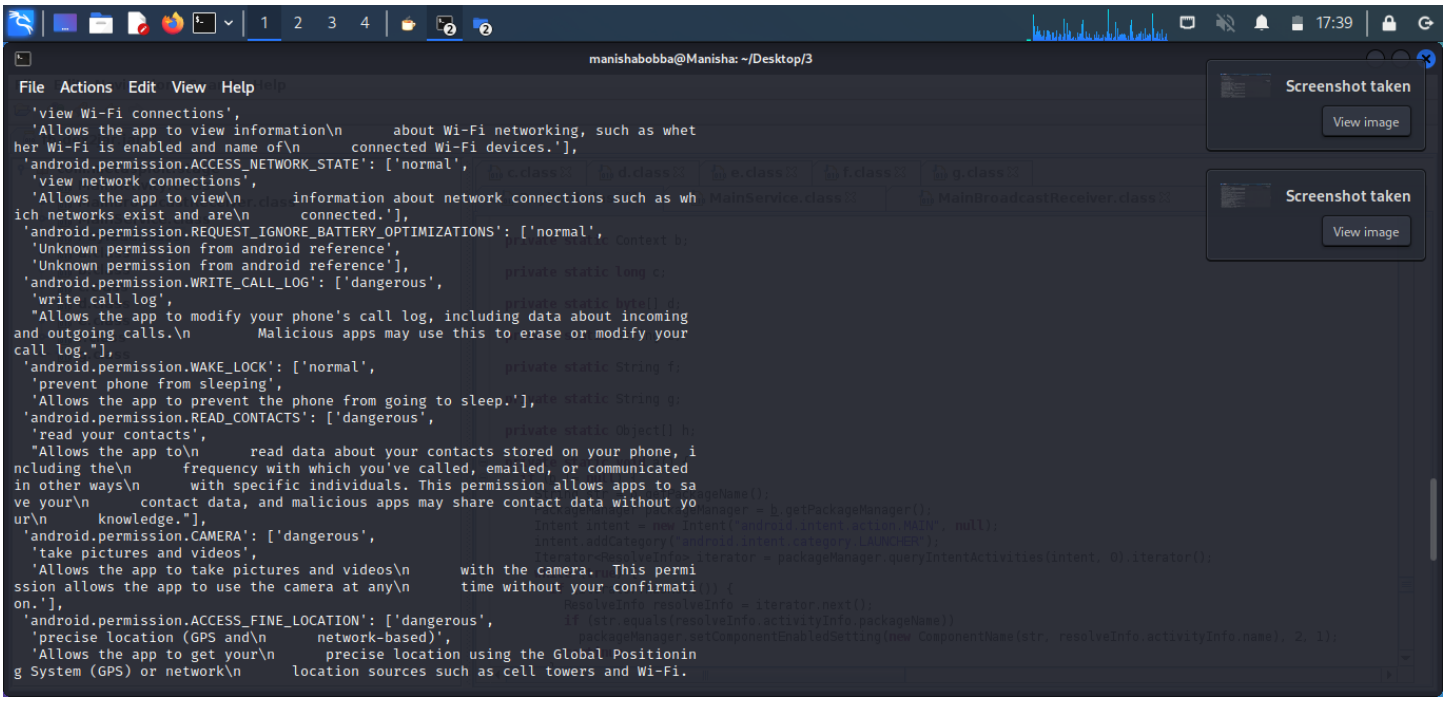
These are the activities that are listed in the APK file.

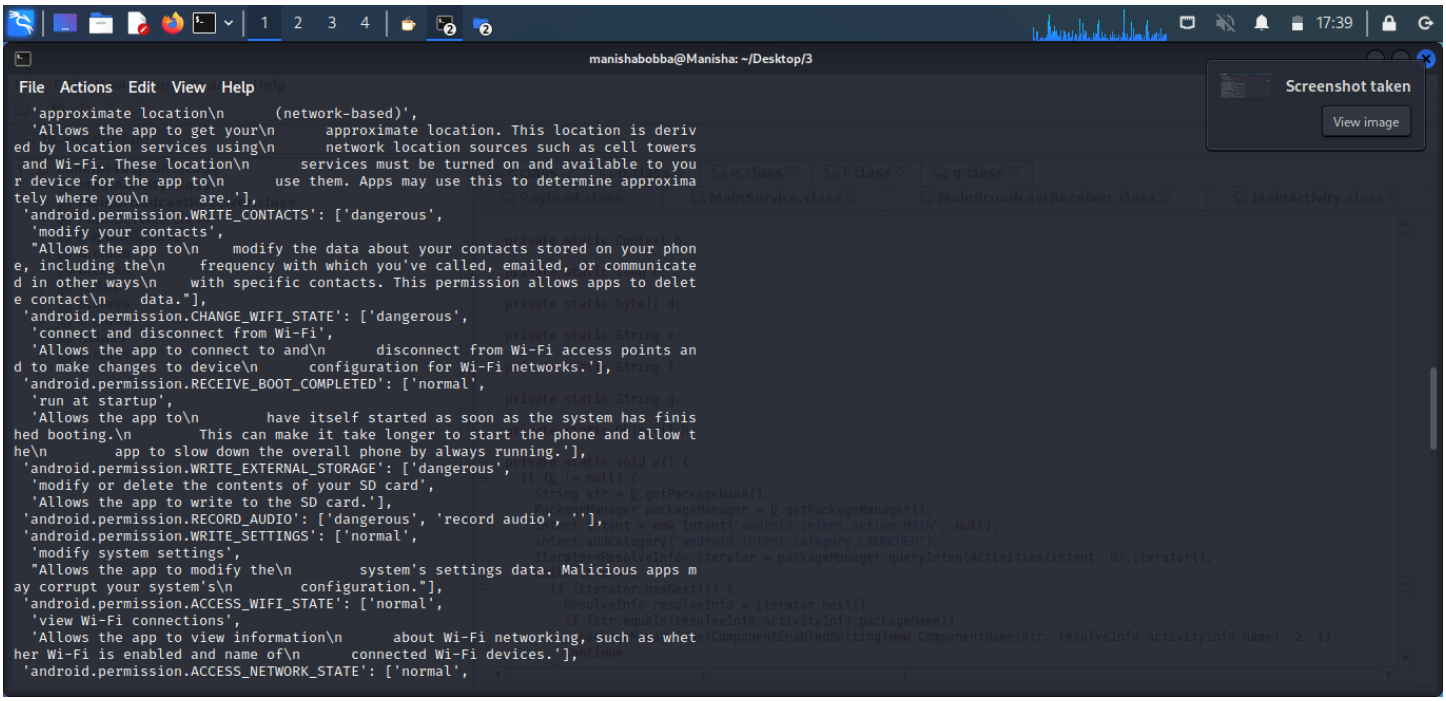
get_activities() is the method used to list the activities of the APK file.

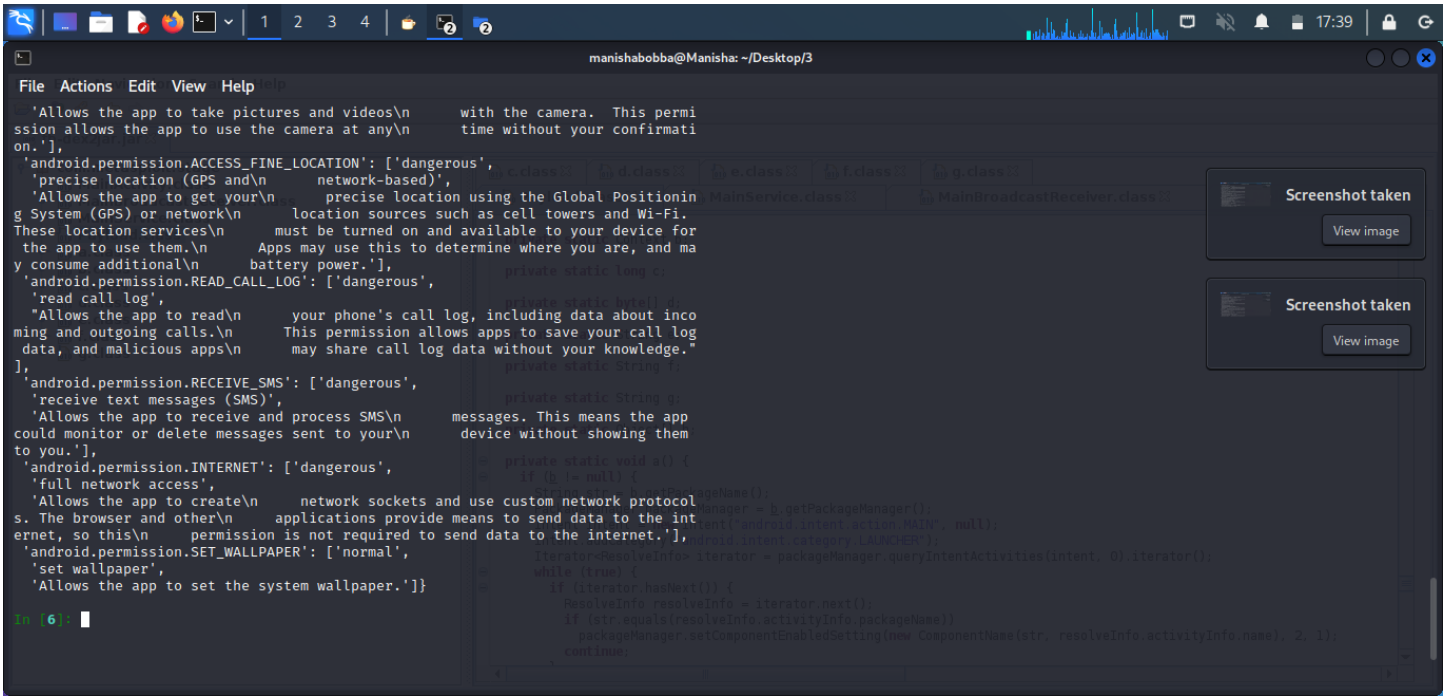
```
In [4]: a.get_activities()
Out[4]: ['com.metasploit.stage.MainActivity']
```

The below screen shots show what does the normal and malicious part of the code will do.









The below listed activities are coming under **dangerous** part of the code.

1. Directly call phone numbers
2. Read phone status and identity
3. Read your text messages (SMS or MMS)
4. Send SMS messages
5. Approximate location
6. Allows app to modify your call log
7. Read your contacts
8. Take pictures and videos
9. Allows the app to get your precise location or GPRS

10. Modify your contacts
11. Connect and disconnect from Wi-Fi
12. Modify or delete the contents of your SD card
13. Receive text messages
14. Full network access

The below listed activities comes under the **normal** part of the code.

1. View network connections
2. Prevent phone from sleeping
3. Run ate startup
4. View Wi-Fi connections
5. Modify system settings
6. Allows the app to take pictures and videos

get_package() is used to get the names of the packages in the APK.

```
In [7]: a.get_package()
Out[7]: 'com.metasploit.stage'
```

get_androidversion_code() is used to get the version code of the APK file that is included in build.gradle or AndroidManifest.xml

```
In [8]: a.get_androidversion_code()
Out[8]: '1'
```

get_androidversion_name() is used to get the version name of the APK file that is included in build.gradle or AndroidManifest.xml

```
In [9]: a.get_androidversion_name()
Out[9]: '1.0'
```

get_min_sdk_version() is used to get the minimum Android SDK version that this package expects to be runnable as specified in the manifest.

```
In [10]: a.get_min_sdk_version()
Out[10]: '10'
```

get_max_sdk_version() is used to get the maximum Android SDK version that this package expects to be runnable as specified in the manifest.

```
In [11]: a.get_max_sdk_version()
```

get_target_sdk_version() is used to get the API Level on which the application is designed to run.

The android system will prevent the user from installing the application if the system's API Level is lower than the values specified in this attribute.

```
In [12]: a.get_target_sdk_version()
Out[12]: '17'
```

get_files() is used to get the classes.

```
In [13]: a.get_files()
Out[13]:
['AndroidManifest.xml',
 'resources.arsc',
 'classes.dex',
 'META-INF/',
 'META-INF/MANIFEST.MF',
 'META-INF/SIGNFILE.SF',
 'META-INF/SIGNFILE.RSA']
```

get_app_icon() is used to get the name of the icon. Here we got nothing as there is no icon present in the APK file.

```
File  Actions  Edit  View  Help
In [14]: a.get_app_icon()
```

get_app_name() is used to get the name of the application.

```
In [15]: a.get_app_name()
Out[15]: 'MainActivity'
```

get_declared_permissions() is used to get the declared permissions. We got nothing as there are no declared permissions.

Return type – list of strings

```
In [16]: a.get_declared_permissions()
Out[16]: []
```

get_declared_permissions_details() is used to get the list of declared permissions.

Return type – dictionary

```
In [17]: a.get_declared_permissions_details()
Out[17]: {}
```

get_dex_names() returns the name of all dex files found in the APK. Thia method onl accounts for "official" dex files in the root directory of apk named classes.dex

Return type – list of strings

```
In [19]: a.get_dex_names()
Out[19]: <filter at 0x7f1269eebc40>
```

get_receivers() returns the android : name attributes of all receivers.

Return type - list of strings

```
In [20]: a.get_receivers()
Out[20]: ['com.metasploit.stage.MainBroadcastReceiver']
```

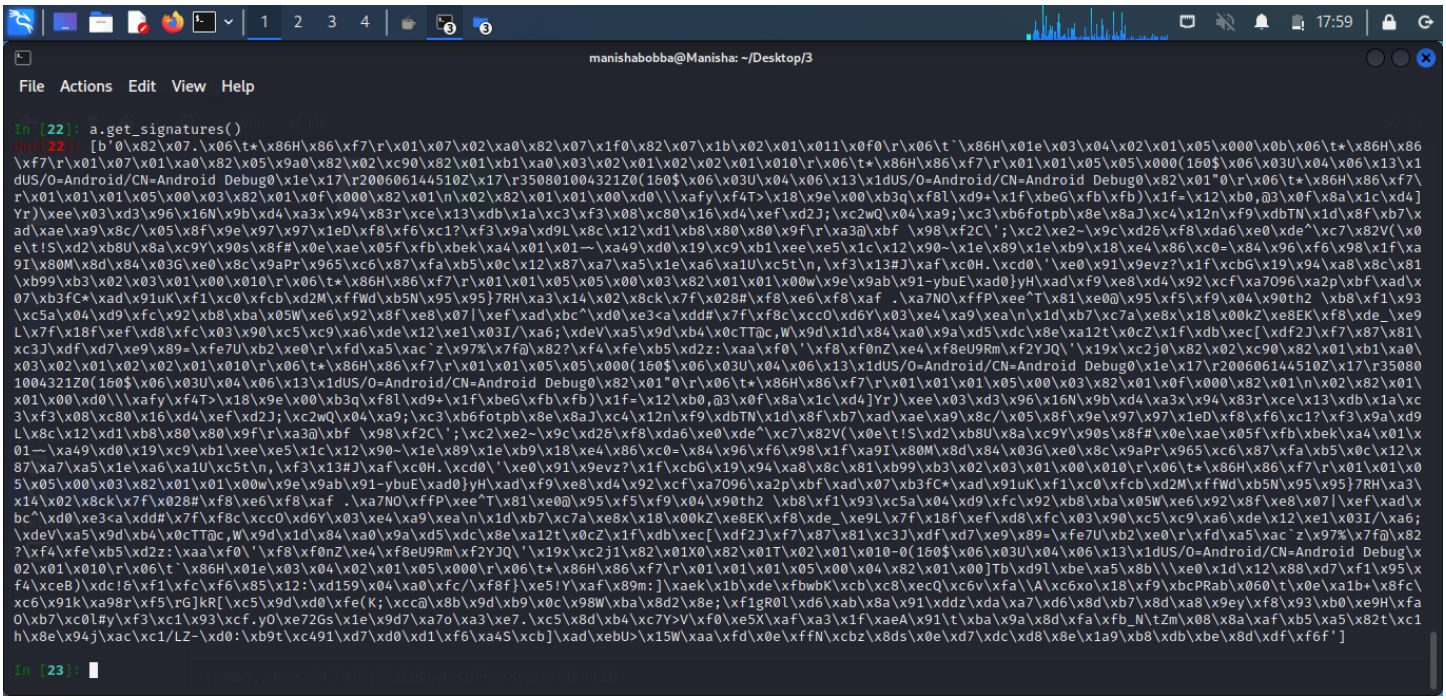
get_services() returns the android : name attributes of all services.

Return type - list of strings

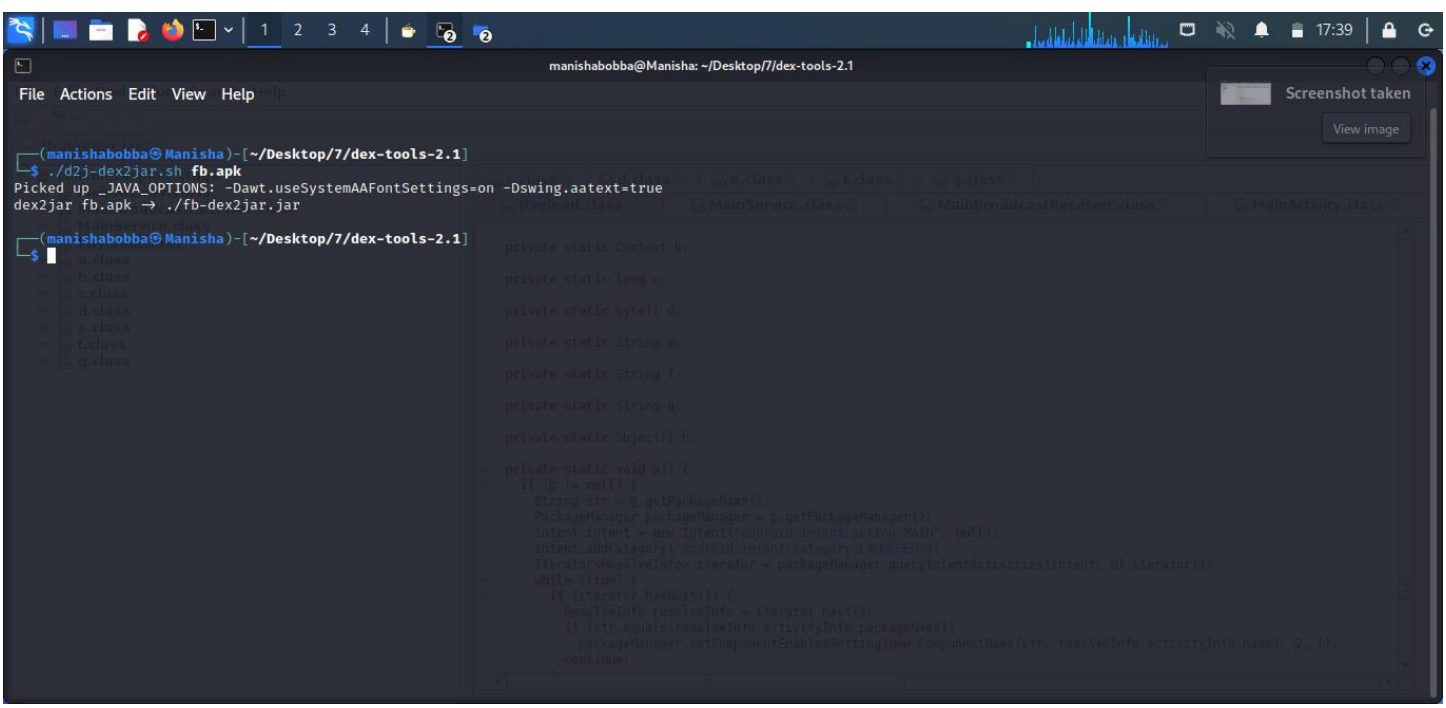
```
In [21]: a.get_services()
Out[21]: ['com.metasploit.stage.MainService']
```

get_signatures() returns a list of the data of the signature files. Only v1 / JAR Signing.

Return type – list of bytes

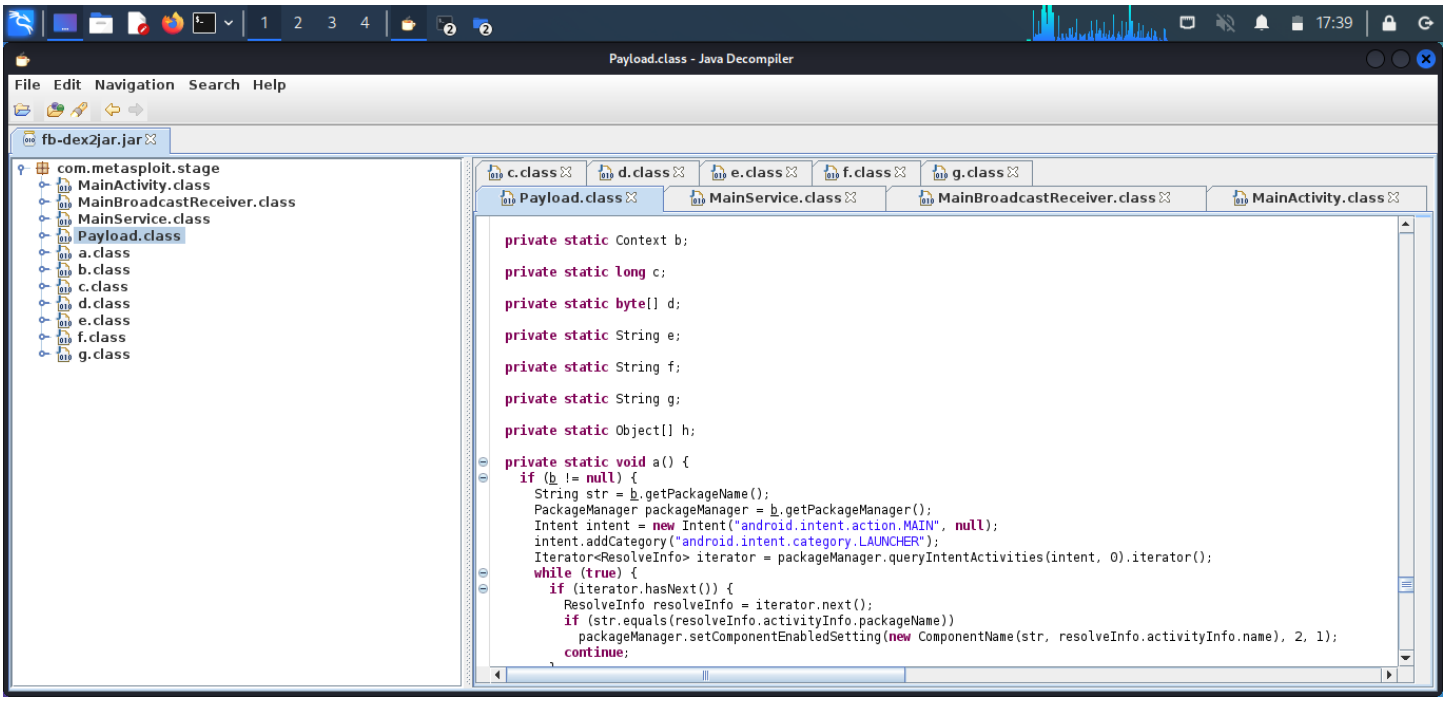


We used dex2jar to convert an APK file to jar file.



We see that the APK is converted to jar file.

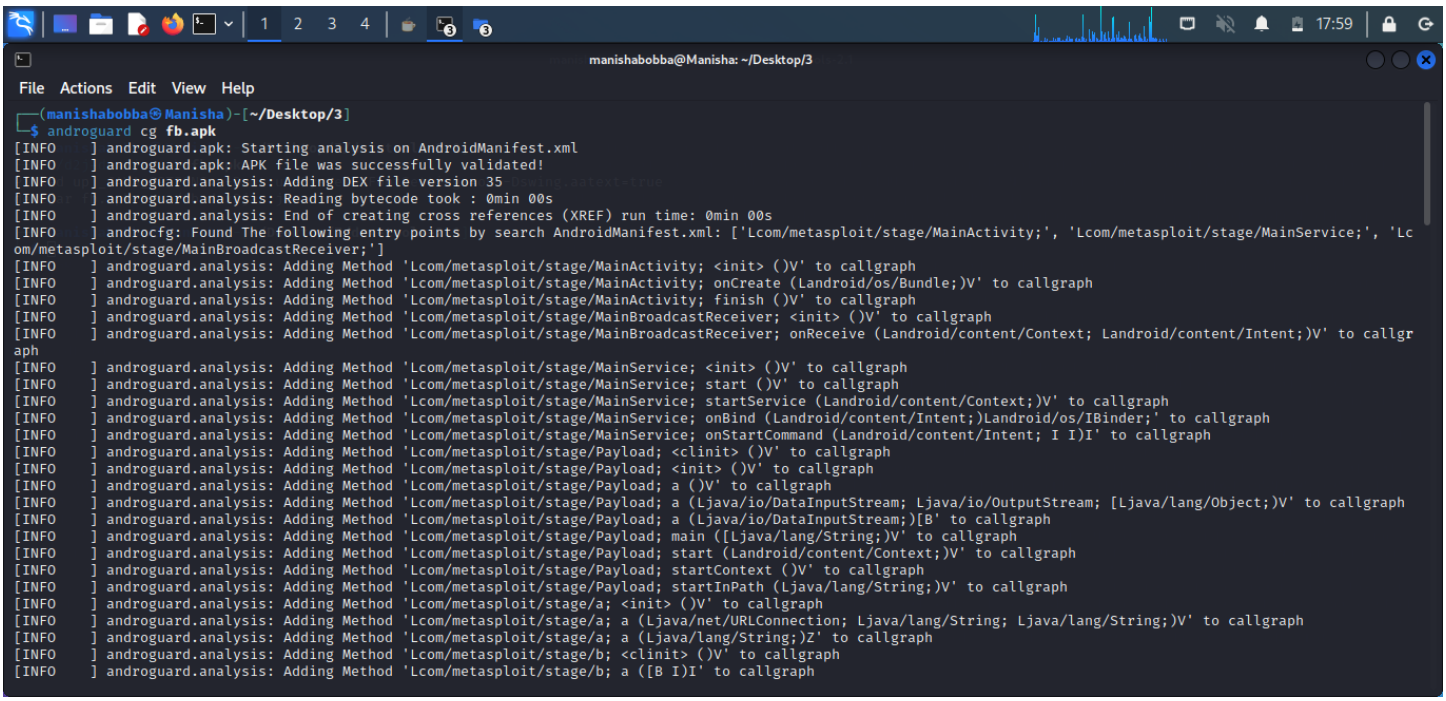
When we open the converted jar file using JD-GUI we can see all the classes and packages of the APK file as below.



The command androguard cg fb.apk is used to create call graph from APK. T

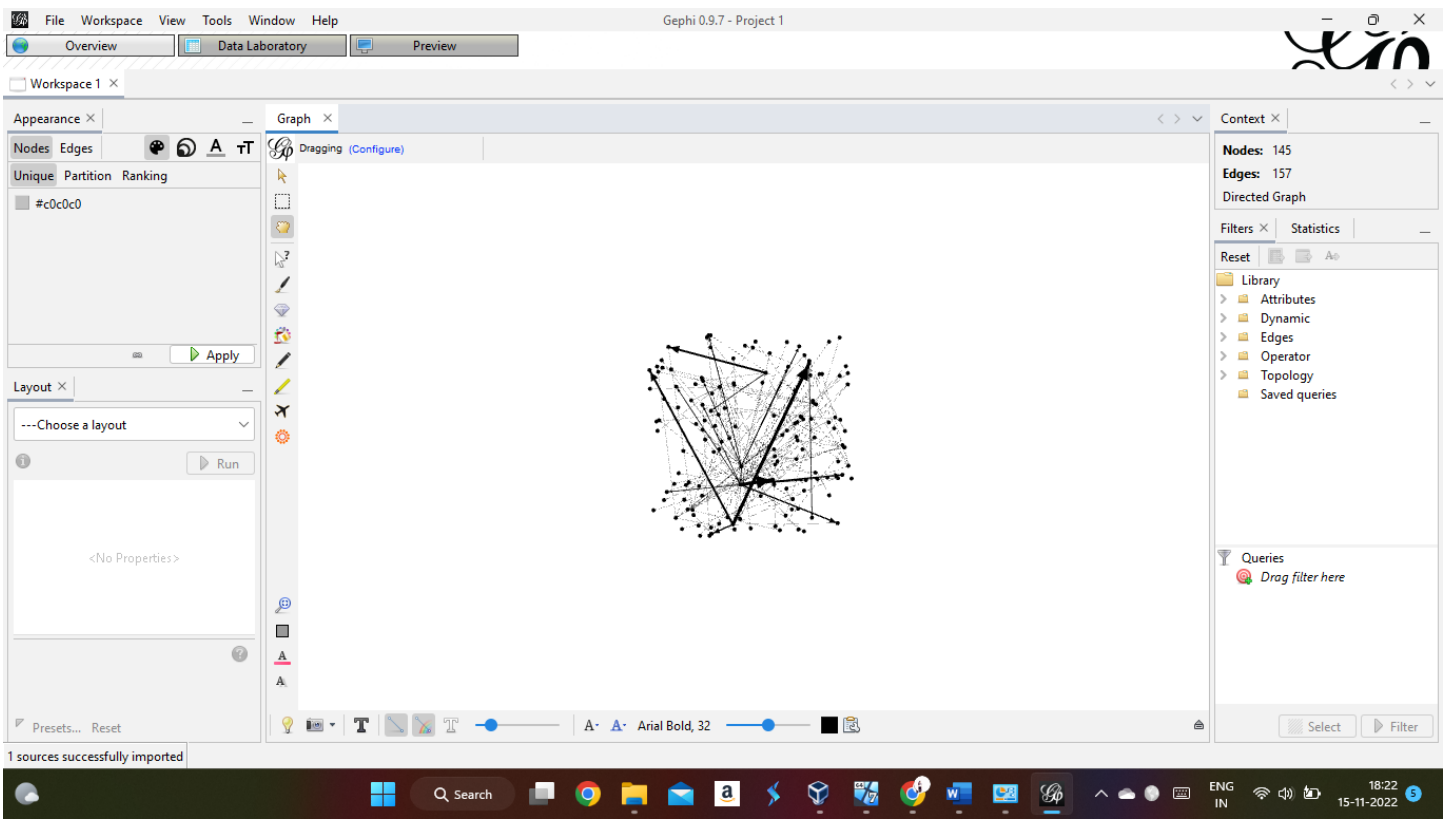
he call graph is constructed from the analysis object and then converted into a networkx MultiDiGraph.

We generated callgraph.gml file which can be opened using Gephi tool.



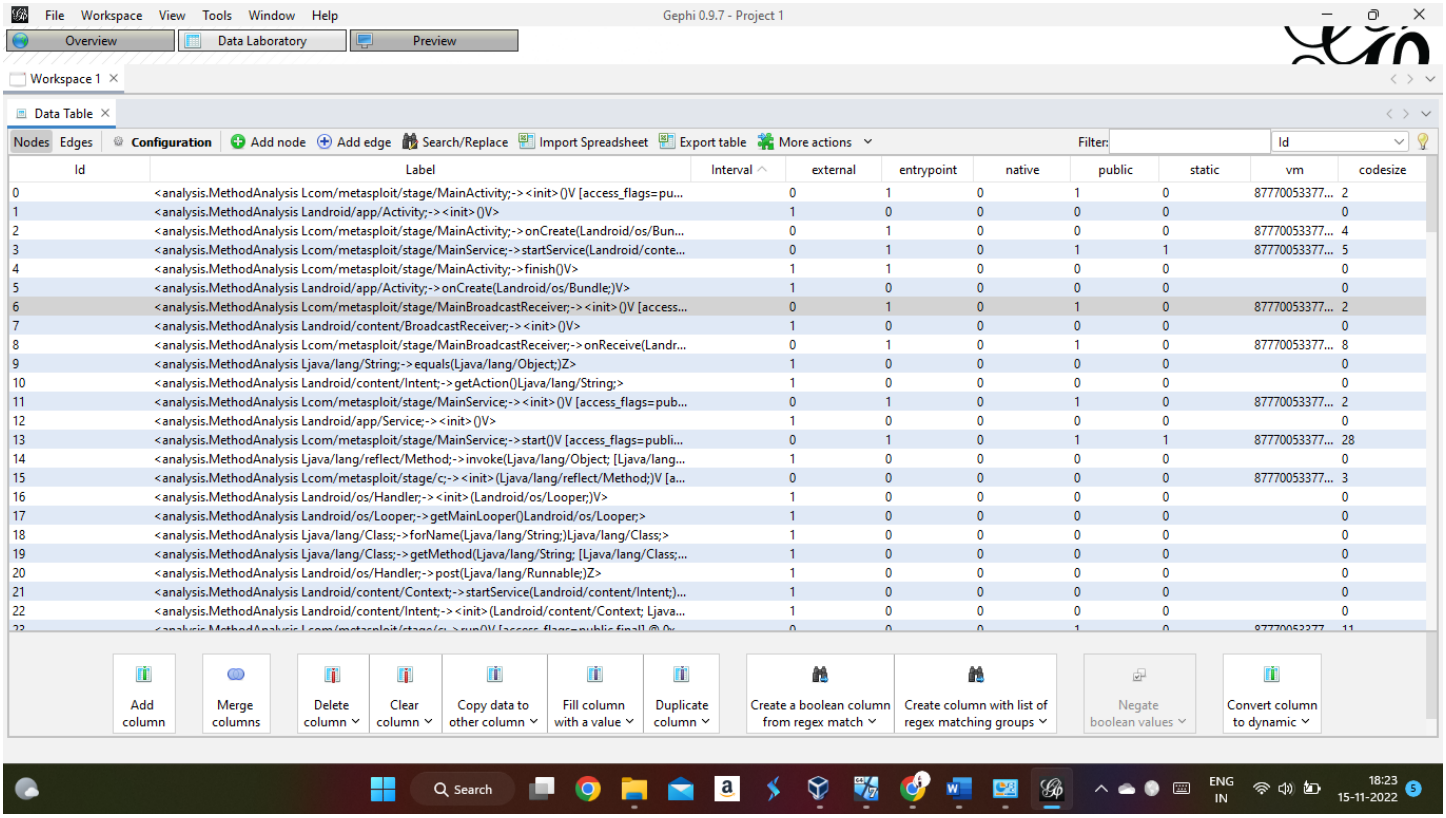
```
manishabobba@Manisha: ~/Desktop/3
File  Actions  Edit  View  Help
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/ServerSocket; <init> (I)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/ServerSocket; accept ()Ljava/net/Socket;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/ServerSocket; close ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/Socket; getInputStream ()Ljava/io/InputStream;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/Socket; getOutputStream ()Ljava/io/OutputStream;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/Socket; <init> (Ljava/lang/String; I)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/io/DataOutputStream; <init> (Ljava/io/OutputStream;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/lang/Thread; sleep (J)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/lang/Thread; <init> ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/URL; <init> (Ljava/lang/String;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/URL; openConnection ()Ljava/net/URLConnection;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/URLConnection; getInputStream ()Ljava/io/InputStream;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/net/URLConnection; addRequestProperty (Ljava/lang/String; Ljava/lang/String;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/io/ByteArrayOutputStream; <init> ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/util/LinkedList; <init> ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/util/concurrent/TimeUnit; toMillis (J)J' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljavax/net/ssl/SSLContext; getInstance (Ljava/lang/String;)Ljavax/net/ssl/SSLContext;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljavax/net/ssl/SSLContext; init ([Ljavax/net/ssl/KeyManager; [Ljavax/net/ssl/TrustManager; Ljava/security/SecureRandom;
)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljavax/net/ssl/SSLContext; getSocketFactory ()Ljavax/net/ssl/SSLSocketFactory;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/SecureRandom; <init> ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljavax/net/ssl/HttpsURLConnection; setSSLSocketFactory (Ljavax/net/ssl/SSLSocketFactory;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljavax/net/ssl/HttpsURLConnection; setHostnameVerifier (Ljavax/net/ssl/HostnameVerifier;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/cert/CertificateException; <init> ()V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/cert/CertificateException; <init> (Ljava/lang/String;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/cert/CertificateException; <init> (Ljava/lang/Throwable;)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/MessageDigest; getInstance (Ljava/lang/String;)Ljava/security/MessageDigest;' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/MessageDigest; update ([B)V' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/MessageDigest; digest ()[B' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/security/cert/X509Certificate; getEncoded ()[B' to callgraph
[INFO    ] androguard.analysis: Adding Method 'Ljava/util/Arrays; equals ([B [B)Z' to callgraph

┌──(manishabobba㉿Manisha)-[~/Desktop/3]
└─$
```

We can see the graph as below in Gephi from the file generated from the above steps.



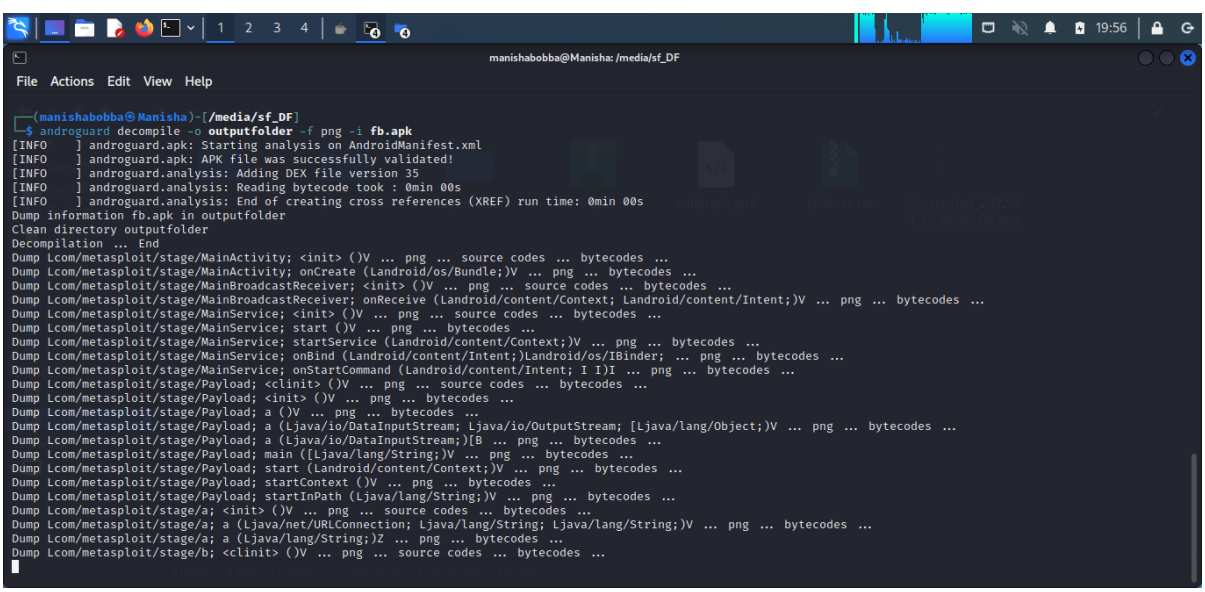
These are all the nodes and edges that show the attributes of every node or edge as table columns of the APK file in Gephi tool.

Here we are showing the methods, label and node ID's.



Id	Label	Interval	external	entrypoint	native	public	static	vm	codesize
0	<analysis.MethodAnalysis Lcom/metasploit/stage/MainActivity;-><init>()V [access_flags=pu...	0	1	0	1	0		87770053377...	2
1	<analysis.MethodAnalysis Landroid/app/Activity;-><init>()V>	1	0	0	0	0			0
2	<analysis.MethodAnalysis Lcom/metasploit/stage/MainActivity;->onCreate(Landroid/os/Bun...	0	1	0	0	0		87770053377...	4
3	<analysis.MethodAnalysis Lcom/metasploit/stage/MainService;->startService(Landroid/conte...	0	1	0	1	1		87770053377...	5
4	<analysis.MethodAnalysis Lcom/metasploit/stage/MainActivity;->finish()V>	1	1	0	0	0			0
5	<analysis.MethodAnalysis Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V>	1	0	0	0	0			0
6	<analysis.MethodAnalysis Lcom/metasploit/stage/MainBroadcastReceiver;-><init>()V [access...	0	1	0	1	0		87770053377...	2
7	<analysis.MethodAnalysis Landroid/content/BroadcastReceiver;-><init>()V>	1	0	0	0	0			0
8	<analysis.MethodAnalysis Lcom/metasploit/stage/MainBroadcastReceiver;->onReceive(Landr...	0	1	0	1	0		87770053377...	8
9	<analysis.MethodAnalysis Ljava/lang/String;->equals(Ljava/lang/Object;)Z>	1	0	0	0	0			0
10	<analysis.MethodAnalysis Landroid/content/Intent;->getAction()Ljava/lang/String;>	1	0	0	0	0			0
11	<analysis.MethodAnalysis Lcom/metasploit/stage/MainService;-><init>()V [access_flags=pub...	0	1	0	1	0		87770053377...	2
12	<analysis.MethodAnalysis Landroid/app/Service;-><init>()V>	1	0	0	0	0			0
13	<analysis.MethodAnalysis Lcom/metasploit/stage/MainService;->start()V [access_flags=publi...	0	1	0	1	1		87770053377...	28
14	<analysis.MethodAnalysis Ljava/lang/reflect/Method;->invoke(Ljava/lang/Object; [Ljava/lang...	1	0	0	0	0			0
15	<analysis.MethodAnalysis Lcom/metasploit/stage/c;-><init>(Ljava/lang/reflect/Method;)V [a...	0	0	0	0	0		87770053377...	3
16	<analysis.MethodAnalysis Landroid/os/Handler;-><init>(Landroid/os/Looper;)V>	1	0	0	0	0			0
17	<analysis.MethodAnalysis Landroid/os/Looper;->getMainLooper()Landroid/os/Looper;>	1	0	0	0	0			0
18	<analysis.MethodAnalysis Ljava/lang/Class;->forName(Ljava/lang/String;)Ljava/lang/Class;>	1	0	0	0	0			0
19	<analysis.MethodAnalysis Ljava/lang/Class;->getMethod(Ljava/lang/String; [Ljava/lang/Class;...	1	0	0	0	0			0
20	<analysis.MethodAnalysis Landroid/os/Handler;->post(Ljava/lang/Runnable;)Z>	1	0	0	0	0			0
21	<analysis.MethodAnalysis Landroid/content/Context;->startService(Landroid/content/Intent;)...	1	0	0	0	0			0
22	<analysis.MethodAnalysis Landroid/content/Intent;-><init>(Landroid/content/Context; Ljava...	1	0	0	0	0			0

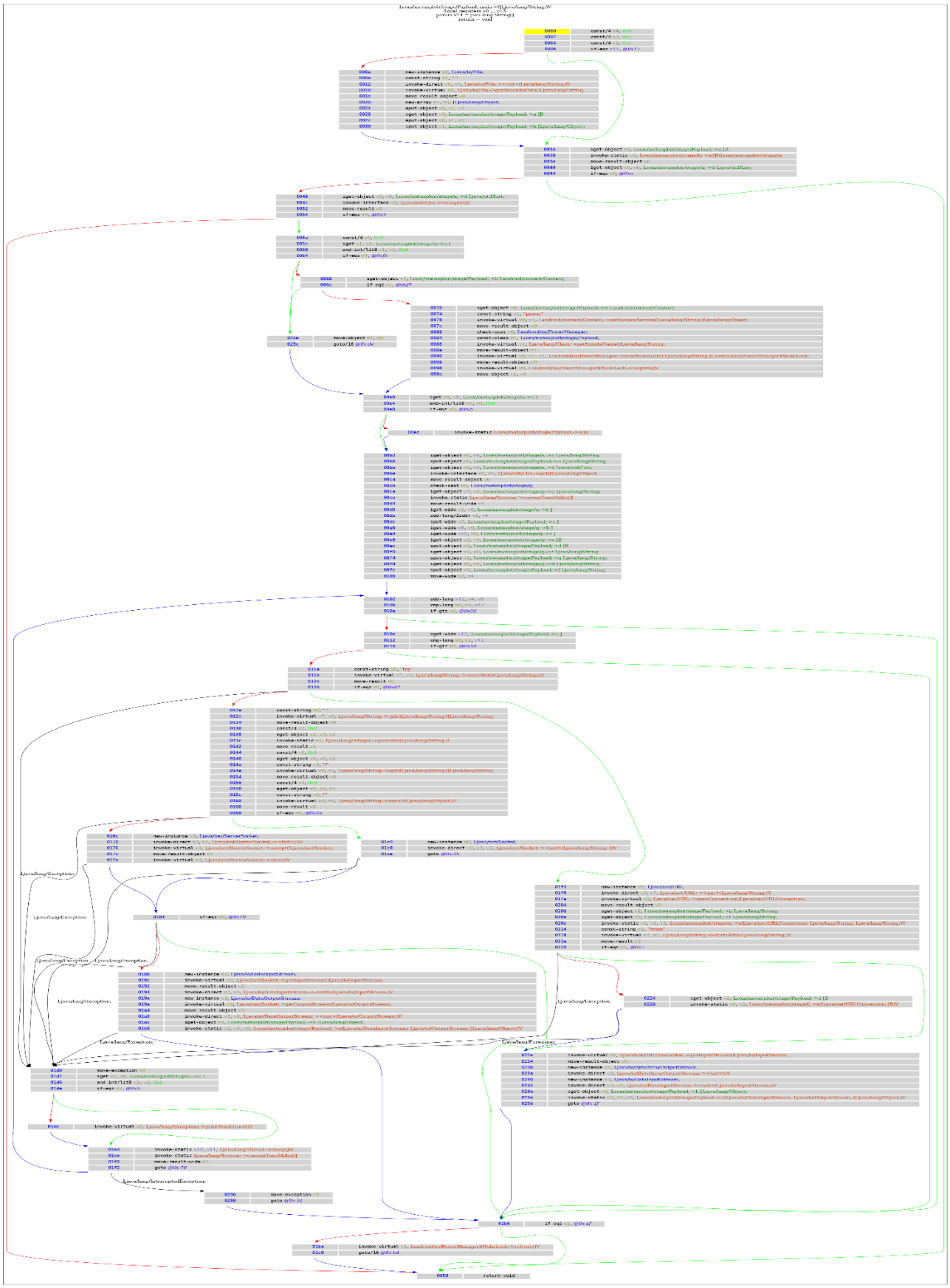
The below command is used to generate the png image of the Control Flow Graphs of the classes of the APK file



We see that the generation of Control Flow Graph is success.

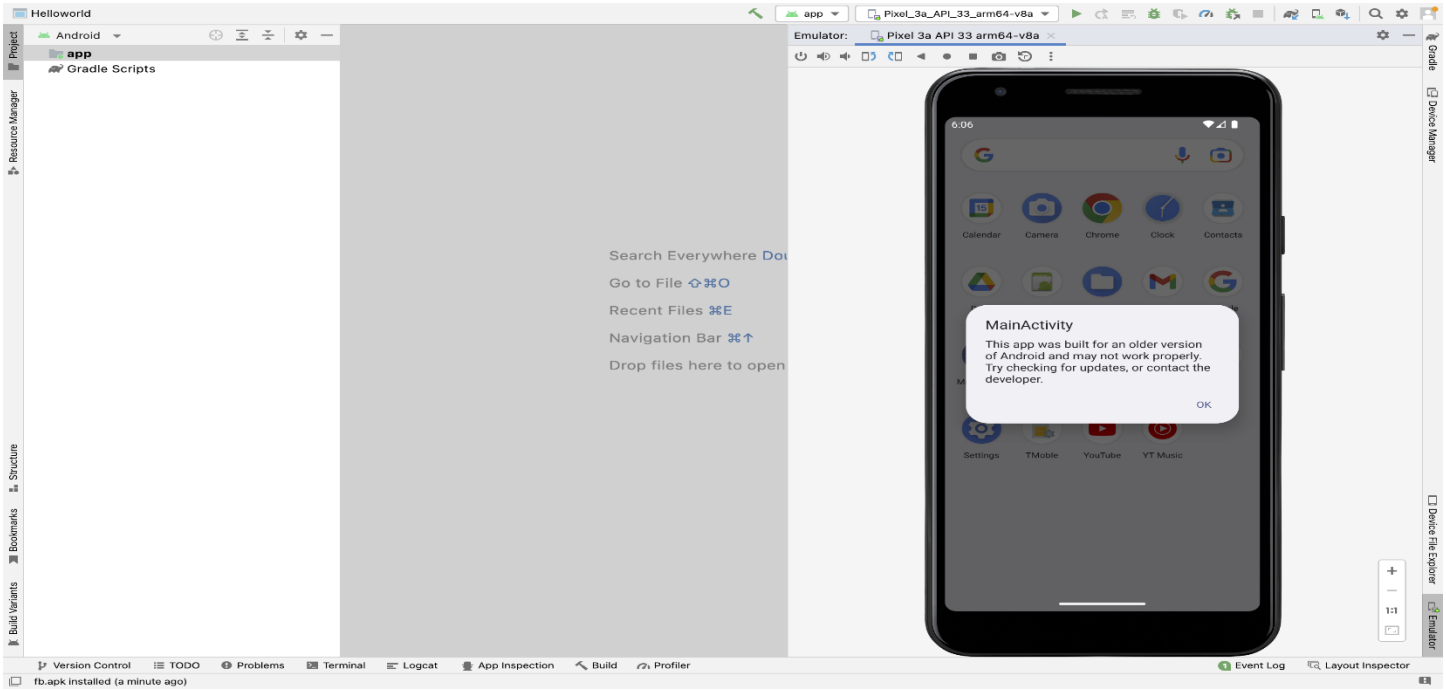
```
manishabobba@Manisha: /media/sf_DF
File  Actions  Edit  View  Help
Dump Lcom/metasploit/stage/MainService; onStartCommand (Landroid/content/Intent; I I)I ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; <clinit> ()V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; <init> ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; a ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; a (Ljava/io/DataInputStream; Ljava/io/OutputStream; [Ljava/lang/Object;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; a (Ljava/io/DataInputStream;)[B ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; main ([Ljava/lang/String;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; start (Landroid/content/Context;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; startContext ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/Payload; startInPath (Ljava/lang/String;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/a; <init> ()V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/a; a (Ljava/net/URLConnection; Ljava/lang/String; Ljava/lang/String;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/a; a (Ljava/lang/String;)Z ... png ... bytecodes ...
Dump Lcom/metasploit/stage/b; <clinit> ()V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/b; a ([B I)I ... png ... bytecodes ...
Dump Lcom/metasploit/stage/b; a ([B)Lcom/metasploit/stage/a; ... png ... bytecodes ...
Dump Lcom/metasploit/stage/b; a ([B I I)Ljava/lang/String; ... png ... bytecodes ...
Dump Lcom/metasploit/stage/b; b ([B I I)[B ... png ... bytecodes ...
Dump Lcom/metasploit/stage/c; <init> (Ljava/lang/reflect/Method;)V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/c; run ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/d; <init> (Ljava/lang/reflect/Method;)V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/d; run ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/e; <init> ()V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/e; run ()V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/f; <init> ([B)V ... png ... source codes ... bytecodes ...
Dump Lcom/metasploit/stage/f; a (Ljava/net/URLConnection; [B)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/f; checkClientTrusted ([Ljava/security/cert/X509Certificate; Ljava/lang/String;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/f; checkServerTrusted ([Ljava/security/cert/X509Certificate; Ljava/lang/String;)V ... png ... bytecodes ...
Dump Lcom/metasploit/stage/f; getAcceptedIssuers ()[Ljava/security/cert/X509Certificate; ... png ... bytecodes ...
Dump Lcom/metasploit/stage/f; verify (Ljava/lang/String; Ljavax/net/ssl/SSLSession;)Z ... png ... bytecodes ...
Dump Lcom/metasploit/stage/g; <init> ()V ... png ... source codes ... bytecodes ...

┌──(manishabobba㉿Manisha)-[/media/sf_DF]
└─$
```

The below graph is the Control Graph of the Payload.class



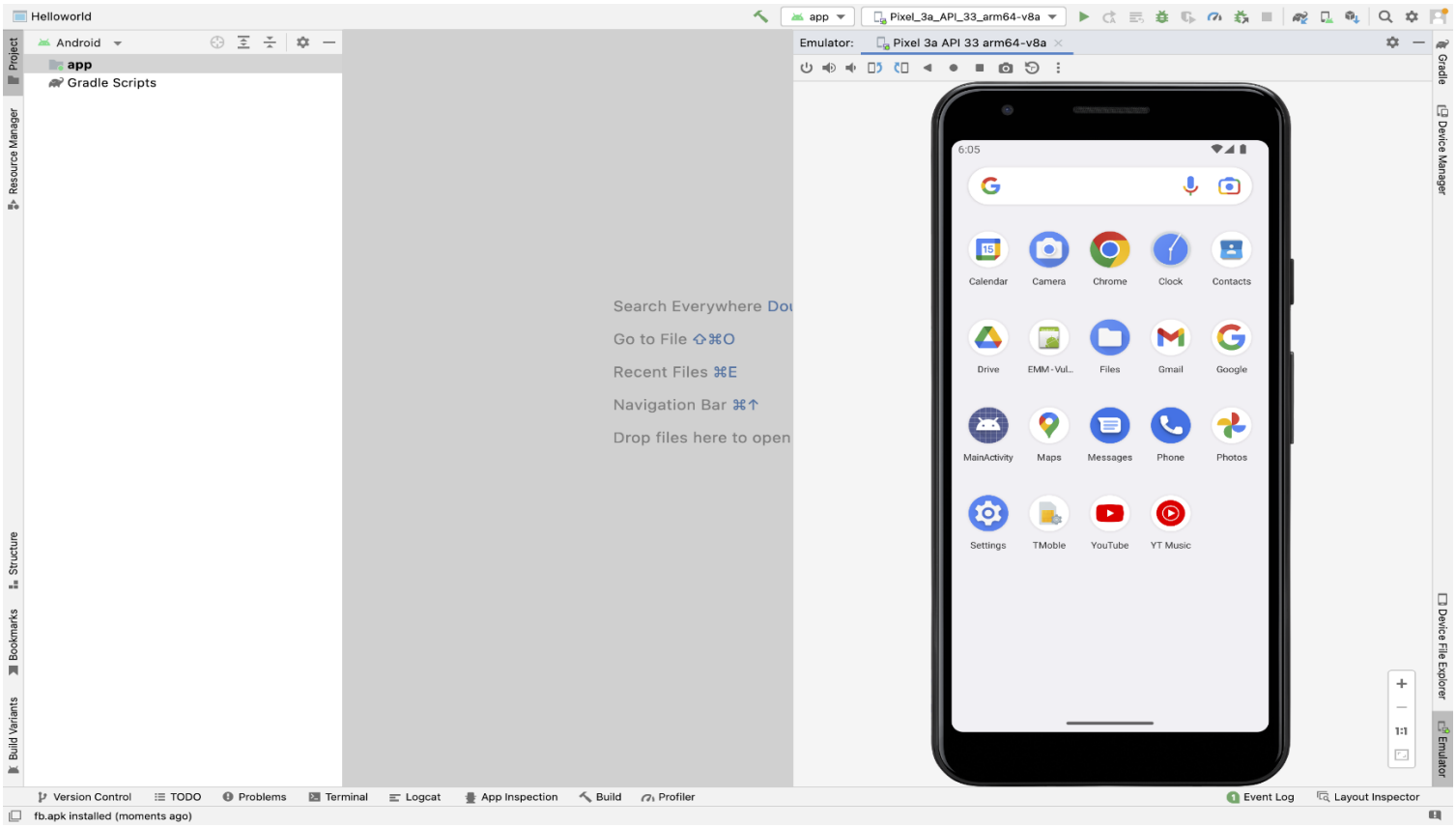
ANALYSIS USING ANDROID STUDIO:

We have tried to install the Malicious APK to the virtual device we observed it faced problems installing in the latest version of the android.



We have downgraded the Android version to 9 then we have successfully installed the application to the virtual device.

The installed APK file name is shown as Mainactivity.



How ever when we tried to open the application it asked for the permissions that the application can access when we grant the permission and click continue to proceed it should open the application how ever here here the application is not moving forward from the permissions page.

So we assume that the application is not installed.