iGNITE Technologies

Android Penetration Testing

MOBSF FRAMEWORK

WWW.HACKINGARTICLES.IN

Contents

Abstract

MobSF is an open-source tool developed by Ajin Abraham that is used for automated analysis of an APK. This is a collection of tools that run under one interface, perform their own individual tasks (like Jadx, apktool etc) and display their results under a common interface. These reports can be downloaded in a PDF format too and give out detailed analysis with necessary screenshots as well. You can download MobSF **here**. In this publication, we'll be walking through the installation phase in Ubuntu OS and guiding you through various options that this tool has to offer.

Installation

To install MobSF, create a directory and follow the commands:

```
git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
cd Mobile-Security-Framework-MobSF
```

```
root@hex:/home/hex/android-toolkit# git clone https://github.com/MobSF/Mobile-Security-Framewo
rk-MobSF.git
Cloning into 'Mobile-Security-Framework-MobSF'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 16859 (delta 0), reused 1 (delta 0), pack-reused 16856
Receiving objects: 100% (16859/16859), 1.09 GiB | 7.64 MiB/s, done.
Resolving deltas: 100% (8022/8022), done.
root@hex:/home/hex/android-toolkit# cd Mobile-Security-Framework-MobSF/
root@hex:/home/hex/android-toolkit/Mobile-Security-Framework-MobSF#
```

We need to install dependencies before we are able to run:

```
apt-get install python3-venv
pip3 install -r requirements.txt
```

```
root@hex:/home/hex/android-toolkit/Mobile-Security-Framework-MobSF# apt-get install python3
-venv
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-venv is already the newest version (3.8.2-0ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@hex:/home/hex/android-toolkit/Mobile-Security-Framework-MobSF# pip3 install -r require
ments.txt
Ignoring waitress: markers 'platform_system == "Windows"' don't match your environment
Requirement already satisfied: Django>=3.1.5 in /usr/local/lib/python3.8/dist-packages (fro
m -r requirements.txt (line 1)) (3.1.7)
Requirement already satisfied: lxml>=4.6.2 in /usr/local/lib/python3.8/dist-packages (from
-r requirements.txt (line 2)) (4.6.2)
```

Once done, we can run the setup file to install MobSF and all the components automatically

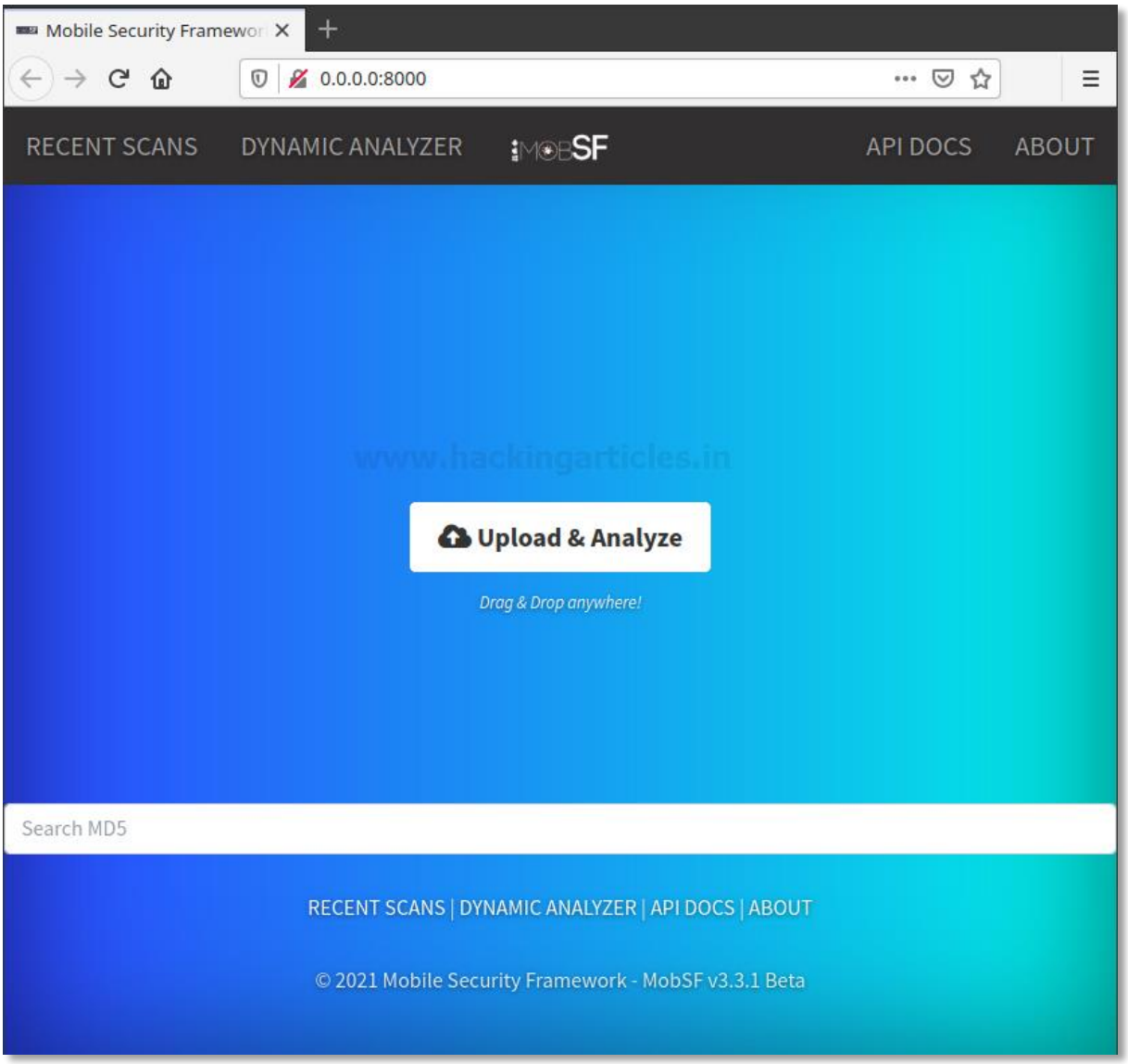
```
./setup.sh
```

```
root@hex:/home/hex/android-toolkit/Mobile-Security-Framework-MobSF# ./setup.sh
[INSTALL] Found Python 3.8.5
pip 21.0.1 from /root/.local/lib/python3.8/site-packages/pip (python 3.8)
[INSTALL] Found pip
Requirement already satisfied: pip in /root/.local/lib/python3.8/site-packages (21.0.1)
[INSTALL] Using python virtualenv
```

Now, to run MobSF we execute the **run.sh** file. As one could interpret from the screenshot below that MobSF would be running on a local server on port 8000.

```
root@hex:/home/hex/android-toolkit/Mobile-Security-Framework-MobSF# ./run.sh
[2021-02-22 21:15:01 +0530] [15422] [INFO] Starting gunicorn 20.0.4
[2021-02-22 21:15:01 +0530] [15422] [INFO] Listening at: http://0.0.0.0:8000 (15422)
[2021-02-22 21:15:01 +0530] [15422] [INFO] Using worker: threads
[2021-02-22 21:15:01 +0530] [15424] [INFO] Booting worker with pid: 15424
```

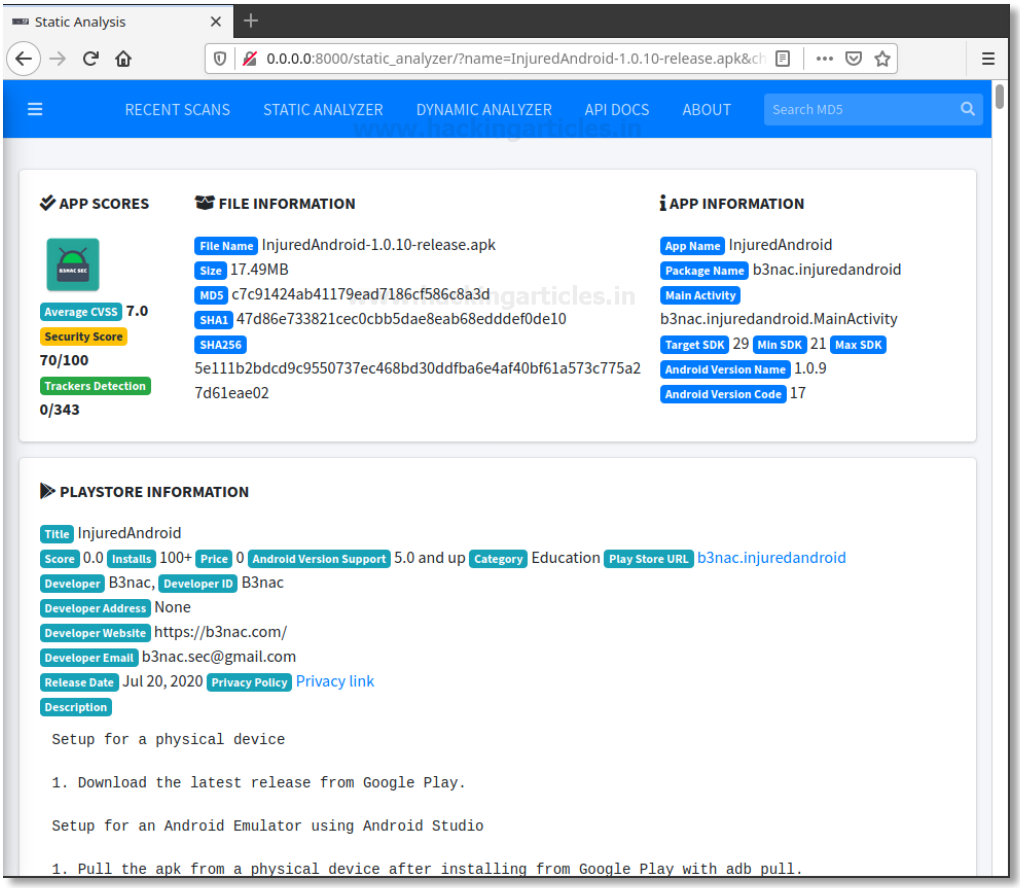
Now let's open the link in the browser and see if MobSF was installed properly or not.



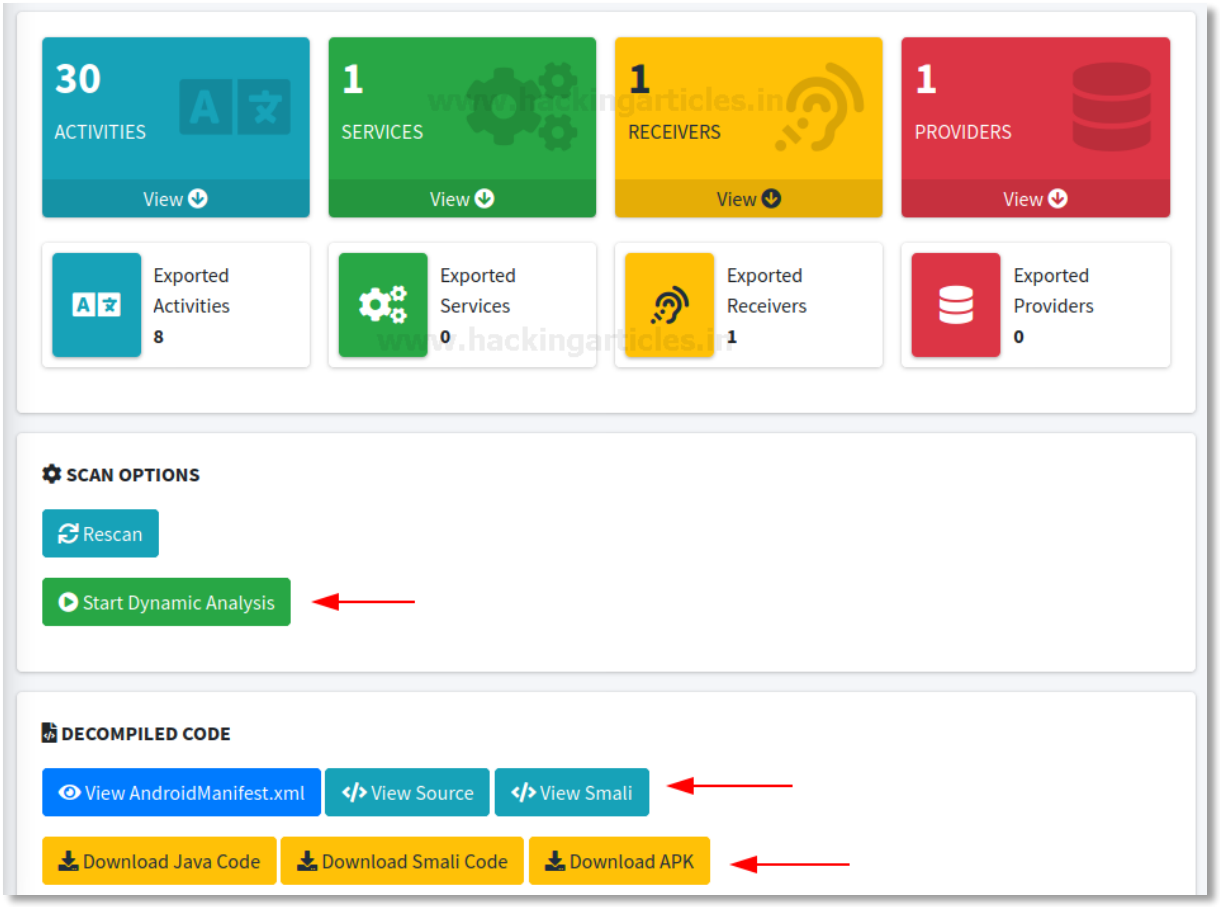
Exploring MobSF

Landing Page

Now that the MobSF is up and running, we can drag a dummy APK (in this case, I'll take InjuredAndroid by Kyle Benac (**here**) into the MobSF interface and see what happens. After waiting for a couple of minutes we could see that static analysis of the APK is done. Now here on the landing page, we can see that a severity score is given. The higher this score the more secure app is. Next, hashes, filename and size of the APK are also given. In the third column in the first row, we can also see the package name, main activity, min SDK version and the application version as well. The description of the application is also given.



After scrolling down a little bit, here's what all we can see:
In small cards, we see different application components
Dynamic analysis option that will help MobSF conduct run time analyses Option to view decompiled code. This is the code that is generated by apktool. Generally, the resources file would also be decoded. It is also possible to view smali code. It makes it easier to segregate and view source code in separate java classes using this.

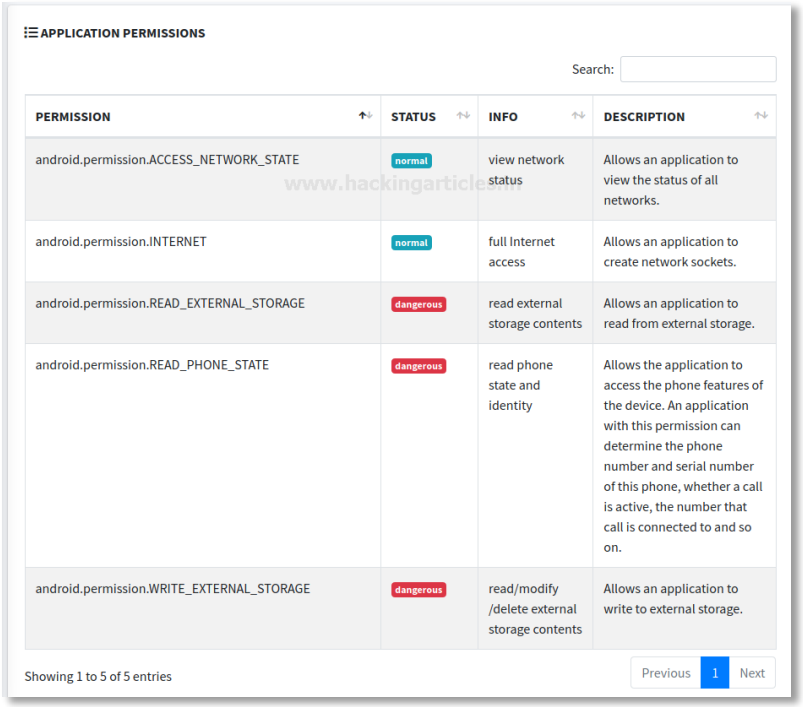


Signer Certificate Analysis

In the certificate column, we can see the signer certificate where one can find important information about the developer, country, state, type of algo, bit size etc.

Application Permissions

Further, we can see all the permissions an application has. There are various permissions that are categorized as dangerous or normal. It is important from a security analyst's point of view to understand which permissions can lead to further damage. For example, if an application has access to external media and stores critical information on the external media it could prove to be dangerous since the files stored on external media are globally readable and writable.



APPLICATION PERMISSIONS

Search:

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.READ_EXTERNAL_STORAGE	dangerous	read external storage contents	Allows an application to read from external storage.
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify /delete external storage contents	Allows an application to write to external storage.

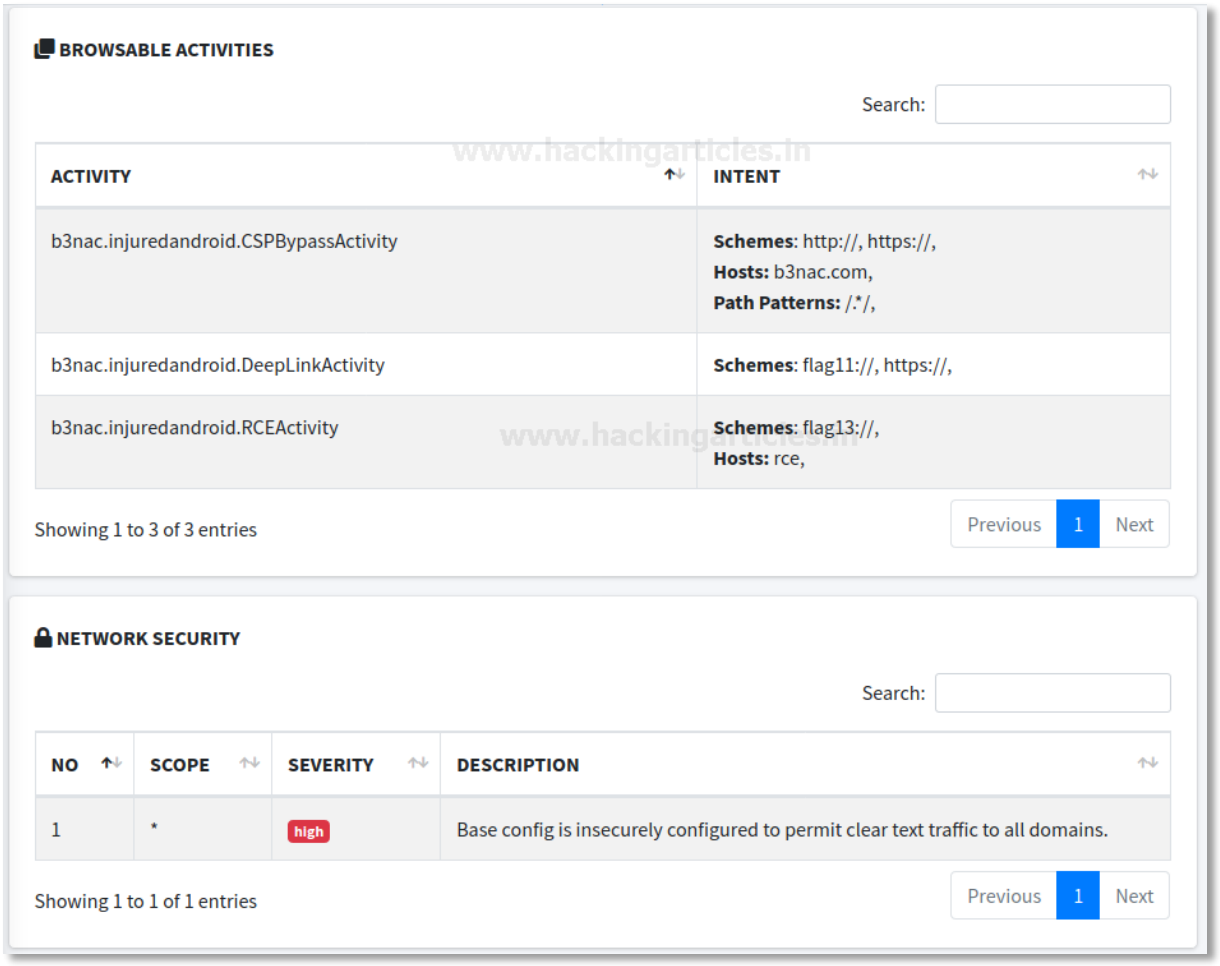
Showing 1 to 5 of 5 entries

Previous 1 Next

Browsable Activities & Network Security Analysis

Next, in the browsable activities section, we can see all the activities that have implemented a deep link schema. Please refer to the article **here** to understand all about deep links, its implementation as well as exploitation.

In the network security section, one can find some details about network security issues related to the application. These issues can lead to critical attacks like MiTM sometimes. For example, in the screenshot below, one can find that the application isn't using the SSL pinning mechanism implemented.



BROWSABLE ACTIVITIES

Search:

ACTIVITY	INTENT
b3nac.injuredandroid.CSPBypassActivity	**Schemes**: http://, https://, **Hosts:** b3nac.com, **Path Patterns:** /.*/,
b3nac.injuredandroid.DeepLinkActivity	**Schemes**: flag11://, https://,
b3nac.injuredandroid.RCEActivity	**Schemes**: flag13://, **Hosts:** rce,

Showing 1 to 3 of 3 entries

Previous 1 Next

NETWORK SECURITY

Search:

NO	SCOPE	SEVERITY	DESCRIPTION
1	*	high	Base config is insecurely configured to permit clear text traffic to all domains.

Showing 1 to 1 of 1 entries

Previous 1 Next

Manifest Analysis

In the next section, MobSF has analysed the manifest file. One can find many folds of information from the android manifest file like which activities are exported, if the app debuggable or not, data schemas etc. For reference look at the screenshot below.

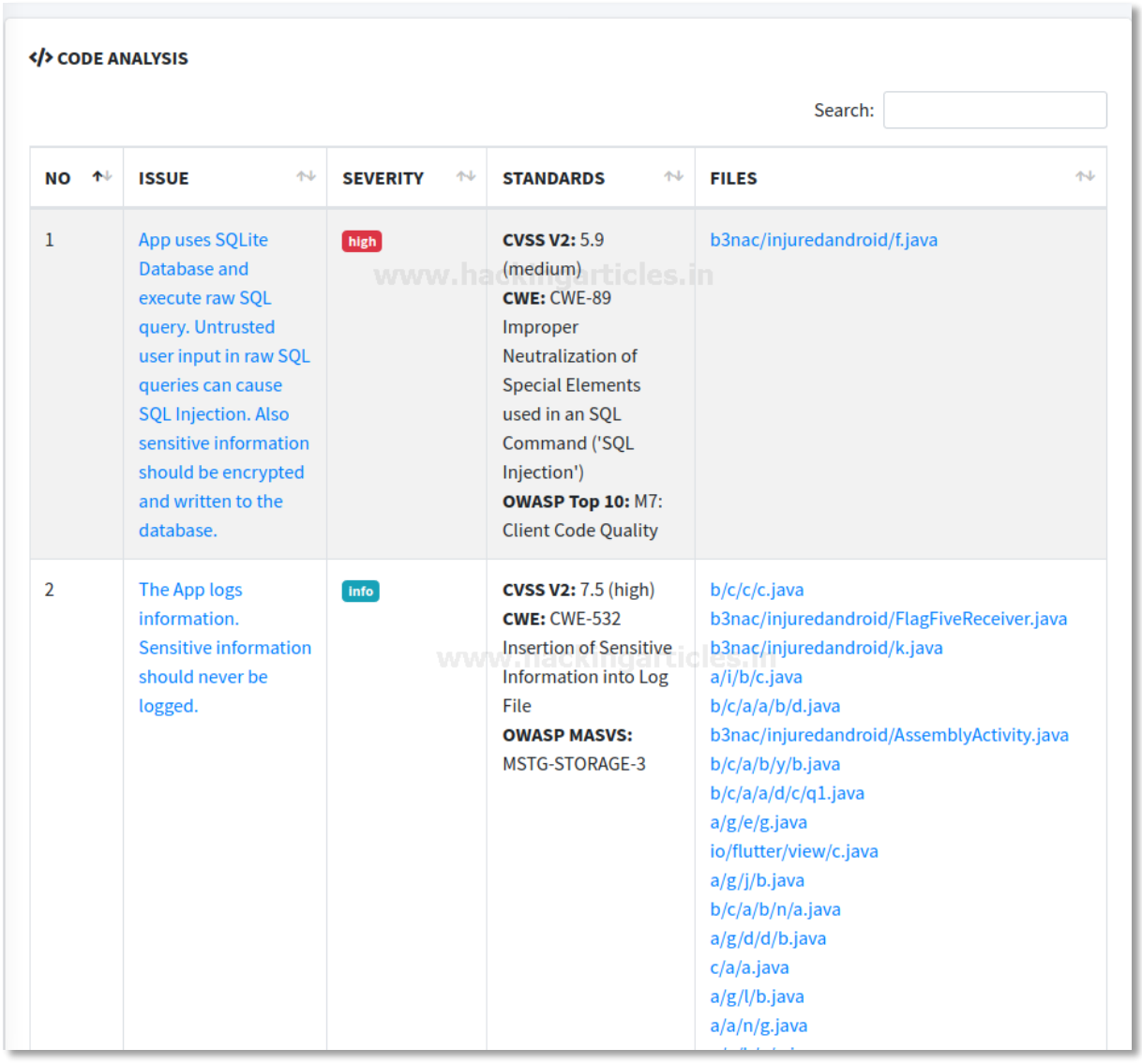
MANIFEST ANALYSIS

Search:

NO	ISSUE	SEVERITY	DESCRIPTION
1	App has a Network Security Configuration [android:networkSecurityConfig=@xml/network_security_config]	info	The Network Security Configuration feature lets apps customize their network security settings in a safe, declarative configuration file without modifying app code. These settings can be configured for specific domains and for a specific app.
2	Application Data can be Backed up [android:allowBackup] flag is missing.	medium	The flag [android:allowBackup] should be set to false. By default it is set to true and allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
3	**Activity** (b3nac.injuredandroid.CSPBypassActivity) is not Protected. An intent-filter exists.	high	An Activity is found to be shared with other apps on the device

Code Analysis

One of the most interesting features of the MobSF tool is the code analysis section. In this section, we can see that MobSF has analysed and compared some behaviour of the application based on industry security standard practices like OWASP MSTG and mapped the vulnerabilities with OWASP Top 10. It is interesting to see CWE mentioned and CVSS score being assigned here which might help various analyst scenarios and help the creation of reports way easier.



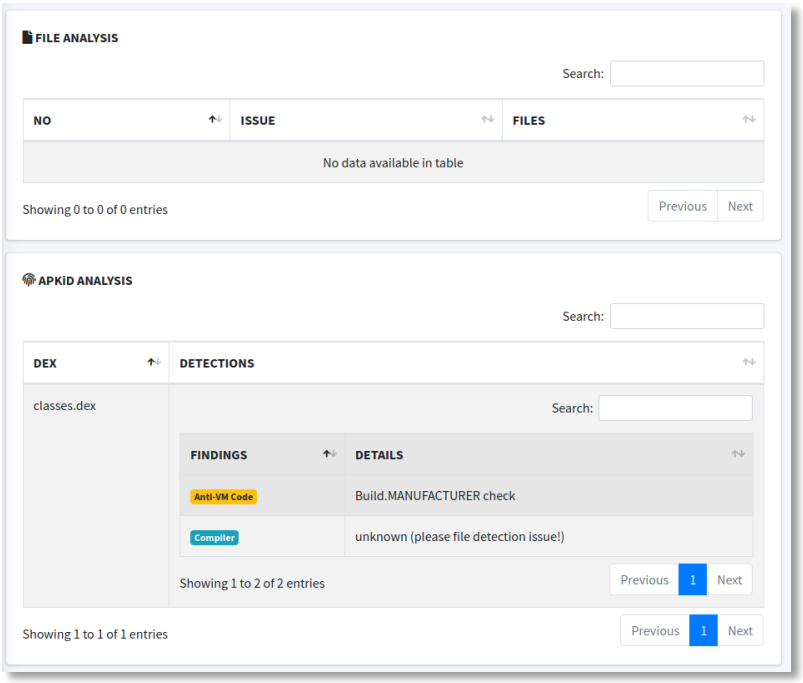
</> CODE ANALYSIS

Search:

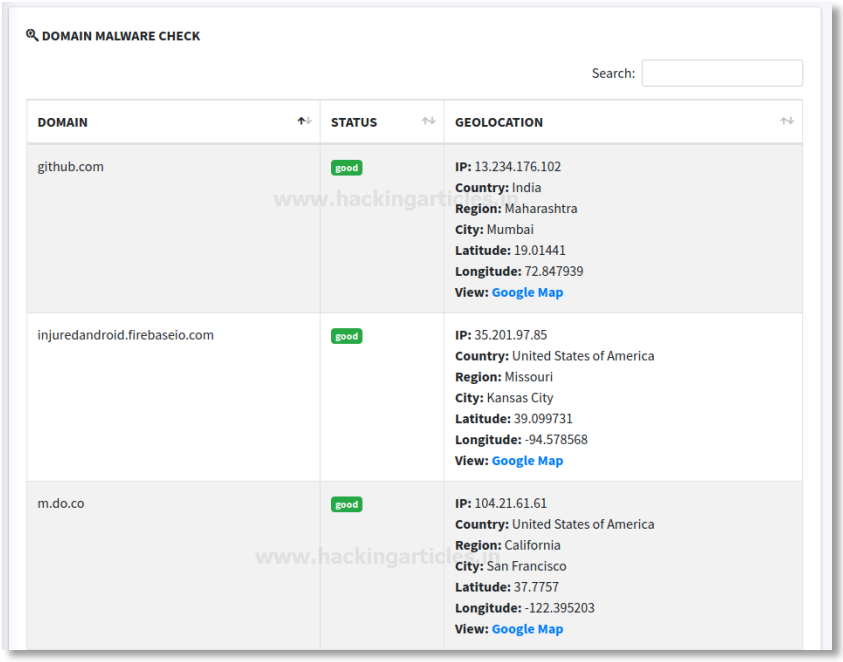
NO	ISSUE	SEVERITY	STANDARDS	FILES
1	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	high	**CVSS V2:** 5.9 (medium) **CWE:** CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') **OWASP Top 10:** M7: Client Code Quality	b3nac/injuredandroid/f.java
2	The App logs information. Sensitive information should never be logged.	info	**CVSS V2:** 7.5 (high) **CWE:** CWE-532 Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	b/c/c/c.java b3nac/injuredandroid/FlagFiveReceiver.java b3nac/injuredandroid/k.java a/i/b/c.java b/c/a/a/b/d.java b3nac/injuredandroid/AssemblyActivity.java b/c/a/b/y/b.java b/c/a/a/d/c/q1.java a/g/e/g.java io/flutter/view/c.java a/g/j/b.java b/c/a/b/n/a.java a/g/d/d/b.java c/a/a.java a/g/l/b.java a/a/n/g.java

Related to Malware Analysis

MobSF also hosts a section where an APKiD analysis is given. APKiD is an open-source tool that is very helpful to identify various packers, compilers, obfuscators etc in android files. It is analogous to PEiD in APK. Here one can see that it has detected an anti-vm code in the APK.



Something related to malware analysis is the domain malware check feature. Here, MobSF is extracting all the URLs/IP addresses that are hard-coded or being used in the application and shows its malware status as well as uses ip2location to give out its geolocation as well.



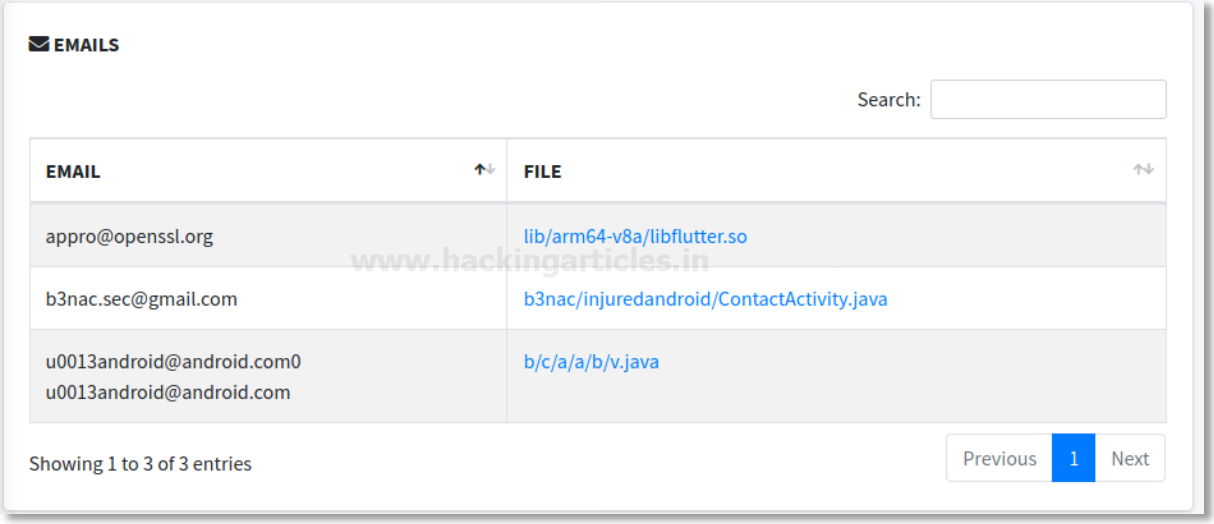
A comprehensive strings analysis is also available. People who are aware of malware analysis know about strings in-depth but for those of you who don't, strings are ASCII and Unicode-printable sequences of characters embedded within a file. Extracting strings can give clues about the program functionality and indicators associated with a suspect binary. For example, if an APK shows something as an output so that stream would be called and hence shown in the strings. This is not the same as strings.xml file. Many times, a third party IP address with which APK is communicating gets visible here. This is essential from a malware analysis point of view.

A STRINGS

id-aes256-wrap
lcircumflexsmall
half4 color = half4(%s, %s, %s, %s);
"abc_searchview_description_query" : "ಪ್ರಶ್ನೆಯನ್ನು ಹುಡುಕಿ"
_input
"common_google_play_services_install_button" : "Инсталиране"
blend_dst_in

experimental
Private_Use_Area
float2 ab = mix(P[0], P[1], T);
Math_Alphanum
Dart_NewStringFromCString
Khojki
"abc_searchview_description_search" : "शोध"
Separator
UNABLE_TO_CREATE_NEW_SECTION
"abc_prepend_shortcut_label" : "Menú +"
"abc_capital_on" : "ಆನ್"
UNKNOWN_EXTENSION_NAME
../../third_party/libcxxabi/src/abort_message.cpp
AHZzsm
[%-8s : sp(%#x) fp(%#x) pc(%#x) %s%s]
"abc_activitychooserview_choose_application" : "Escolher uma aplicação"
GrRenderTargetContext::drawTextureSet
glBindAttribLocation
"common_google_play_services_install_text" : "%1$s нема да се извршува без услугите на Google Play што ги нем
"abc_searchview_description_clear" : "क्वेरी साफ करा"
Show invisible frames in stack traces.
noexcept
"abc_menu_sym_shortcut_label" : "Sym+"
sInt64List.
ENTITIES
"status_bar_notification_info_overflow" : "+999"
null-safety
Surrogate

One can also find hardcoded emails in MobSF. This is all done using the decompiled source code. Often a pentester can find critical email IDs that were being used as a credential on a third party site, say, to access the database.



EMAILS

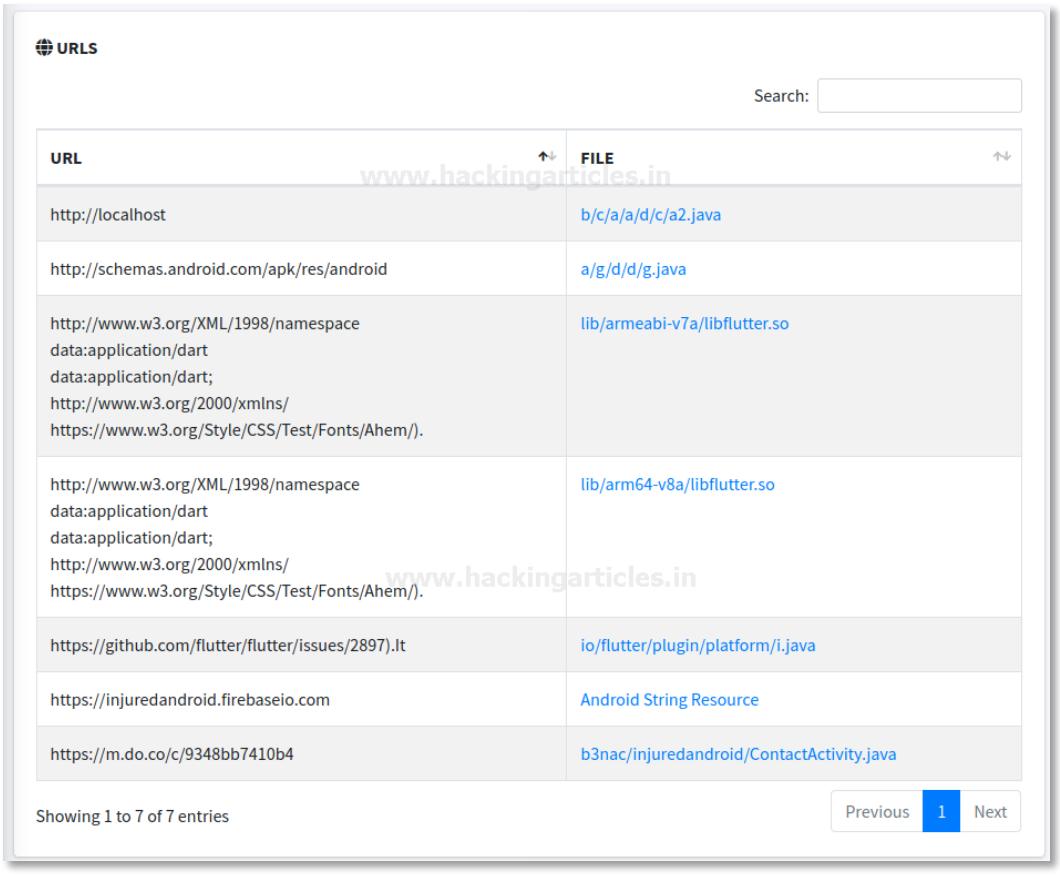
Search:

EMAIL	FILE
appro@openssl.org	lib/arm64-v8a/libflutter.so
b3nac.sec@gmail.com	b3nac/injuredandroid/ContactActivity.java
u0013android@android.com0 u0013android@android.com	b/c/a/a/b/v.java

Showing 1 to 3 of 3 entries

Previous 1 Next

Just like emails, URLs are often found hardcoded as well. One can find juicy URLs that are being used sometimes. Oftentimes analysts find malicious URLs being accessed as well or even a C&C server.



URLS

Search:

URL	FILE
http://localhost	b/c/a/a/d/c/a2.java
http://schemas.android.com/apk/res/android	a/g/d/d/g.java
http://www.w3.org/XML/1998/namespace data:application/dart data:application/dart; http://www.w3.org/2000/xmlns/ https://www.w3.org/Style/CSS/Test/Fonts/Ahem/).	lib/armeabi-v7a/libflutter.so
http://www.w3.org/XML/1998/namespace data:application/dart data:application/dart; http://www.w3.org/2000/xmlns/ https://www.w3.org/Style/CSS/Test/Fonts/Ahem/).	lib/arm64-v8a/libflutter.so
https://github.com/flutter/flutter/issues/2897).It	io/flutter/plugin/platform/i.java
https://injuredandroid.firebaseio.com	Android String Resource
https://m.do.co/c/9348bb7410b4	b3nac/injuredandroid/ContactActivity.java

Showing 1 to 7 of 7 entries

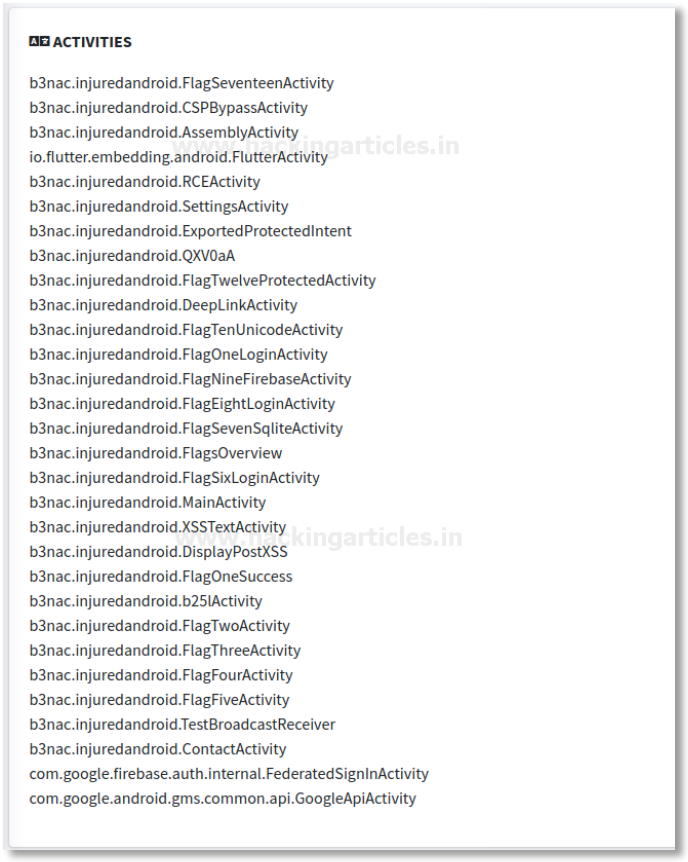
Previous 1 Next

Hardcoded secrets

Oftentimes developers have this habit of storing critical keys like AWS ID and credentials in strings.xml and use an object as a reference in java activity. But doing this doesn't help in any which way since strings.xml can be decoded easily.

Activity Components Present

A list of all the activities present can also be scrolled using MobSF. This gives an insight into the skeleton of the android APK. Also sometimes jadx replaces the real names of the class with some random letter if the developer has applied obfuscation, MobSF can associate its real name too (doesn't happen all the time or in cases of strong obfuscation).



Quite similarly an analyst can also traverse services, broadcast, providers and content receivers along with all the files present in the APK archive to create a map of all the resources present in the application.

SERVICES

com.google.firebase.components.ComponentDiscoveryService

RECEIVERS

b3nac.injuredandroid.FlagFiveReceiver

PROVIDERS

com.google.firebase.provider.FirebaseInitProvider

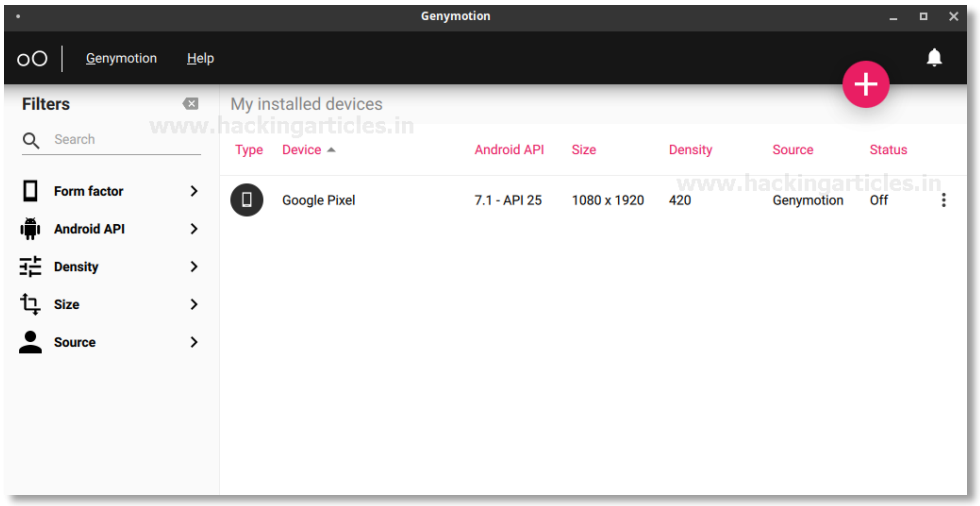
LIBRARIES

FILES

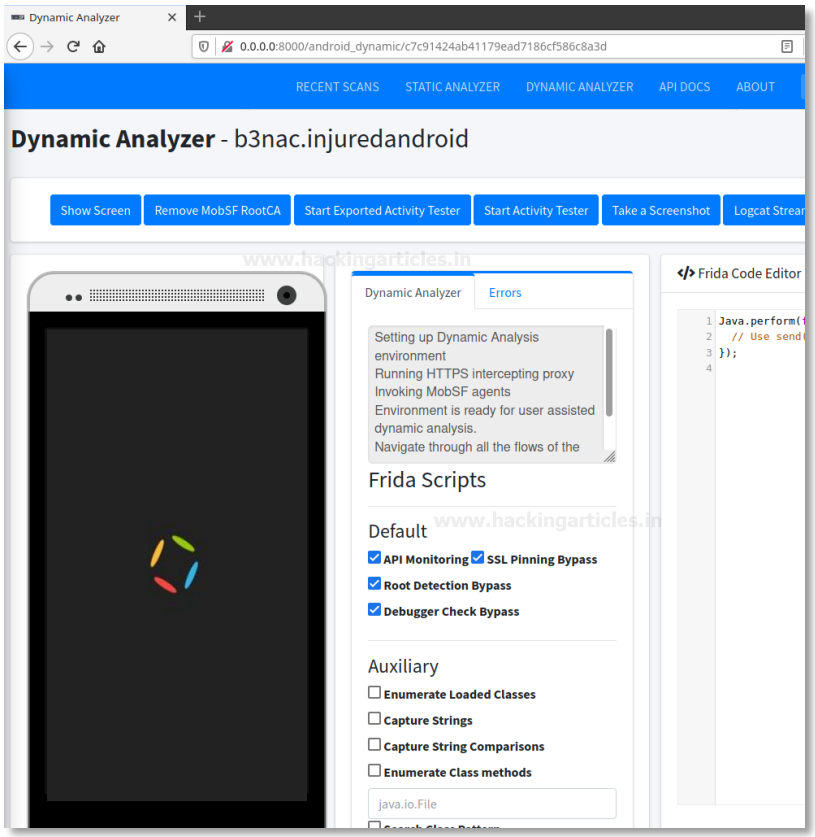
AndroidManifest.xml
META-INF/CERT.RSA
META-INF/CERT.SF
META-INF/MANIFEST.MF
META-INF/androidx.activity_activity.version
META-INF/androidx.appcompat_appcompat-resources.version
META-INF/androidx.appcompat_appcompat.version
META-INF/androidx.arch.core_core-runtime.version
META-INF/androidx.cardview_cardview.version
META-INF/androidx.coordinatorlayout_coordinatorlayout.version
META-INF/androidx.core_core-ktx.version
META-INF/androidx.core_core.version

Dynamic Analyzer

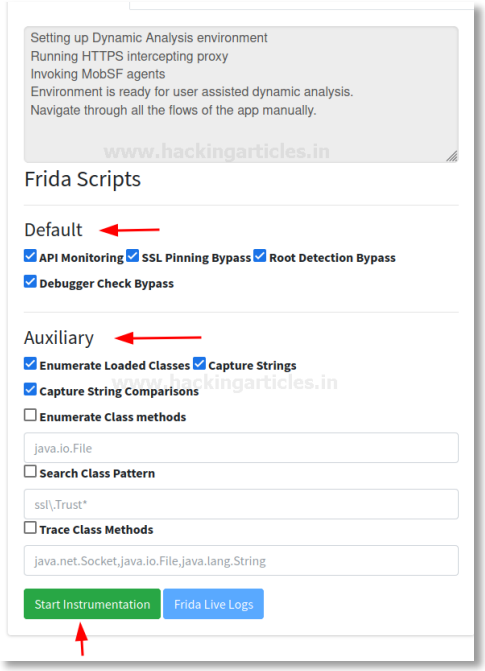
For dynamic analysis, we'd need to fire up android VM in genymotion first. Here I've created an android VM on version 7.1



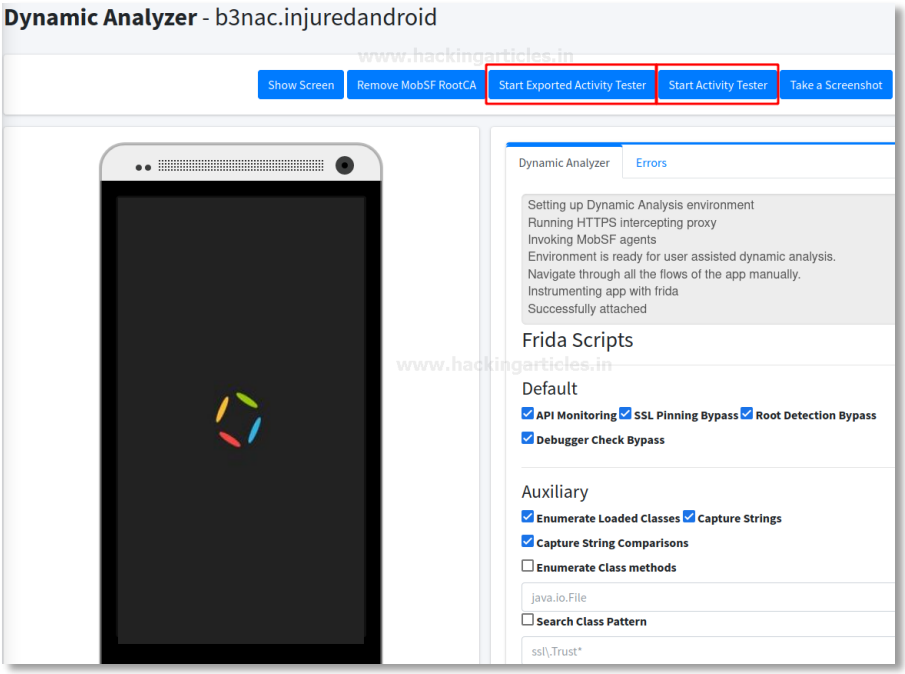
When you press the dynamic analyser option present on the top navigation pane, MobSF will automatically attach itself to VM running if MobSF and genymotion are running on the same base machine. However, if MobSF is in another virtual machine, you might have to attach MobSF agent to genymotion's VM's remote IP and port. Once it is attached, we see the following screen.



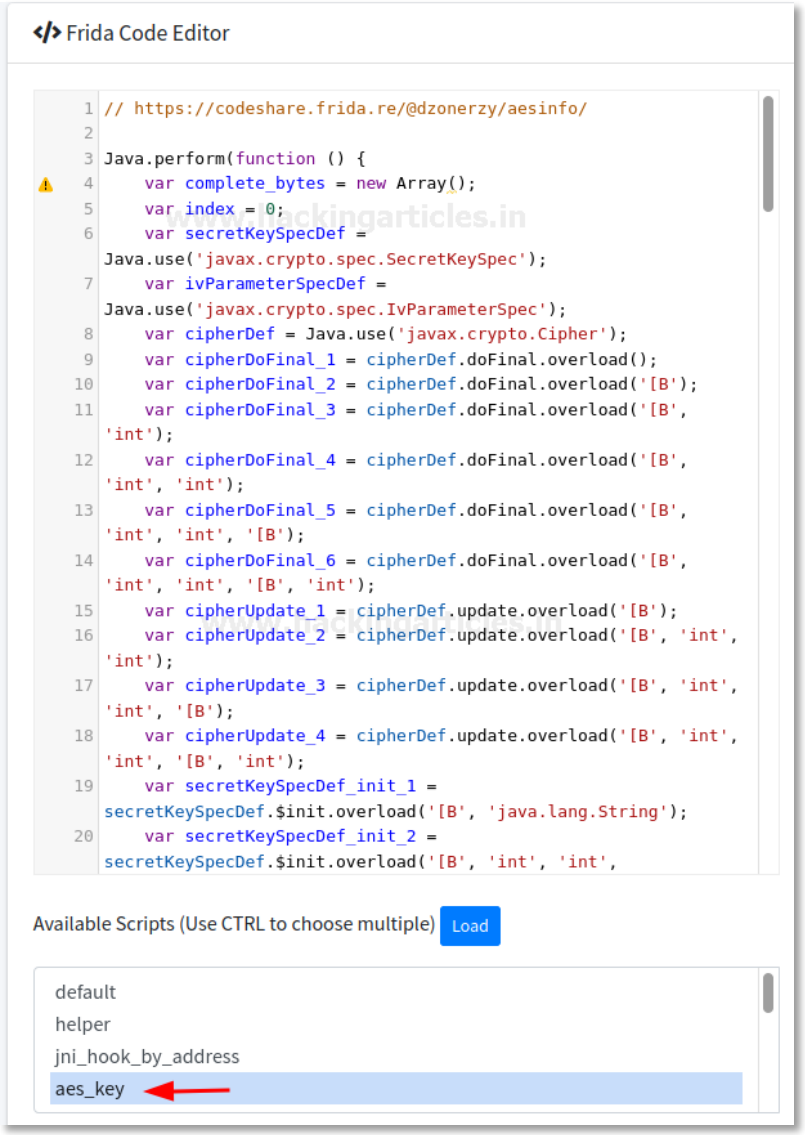
Under the analyzer status bar we can see various default frida scripts available that would check various basic vulnerabilities like SSL Pinning bypass and Root detection checks. If you haven't read about frida, please do so by going here. There are other auxiliary scripts as well that lets an analyst enumerate various classes and also capture string comparisons in real time (again helpful for malware analysts point of view). Then simply click on start instrumentation and the selected scripts will be attached to the application automatically. Hence, if I have selected SSL Pinning bypass script and traffic is getting captured (visible in log or API monitor later) that would mean SSL Pinning has got bypassed.



Now further, to analyse activities for vulnerabilities one can see two buttons on the top for both exported and non exported activities

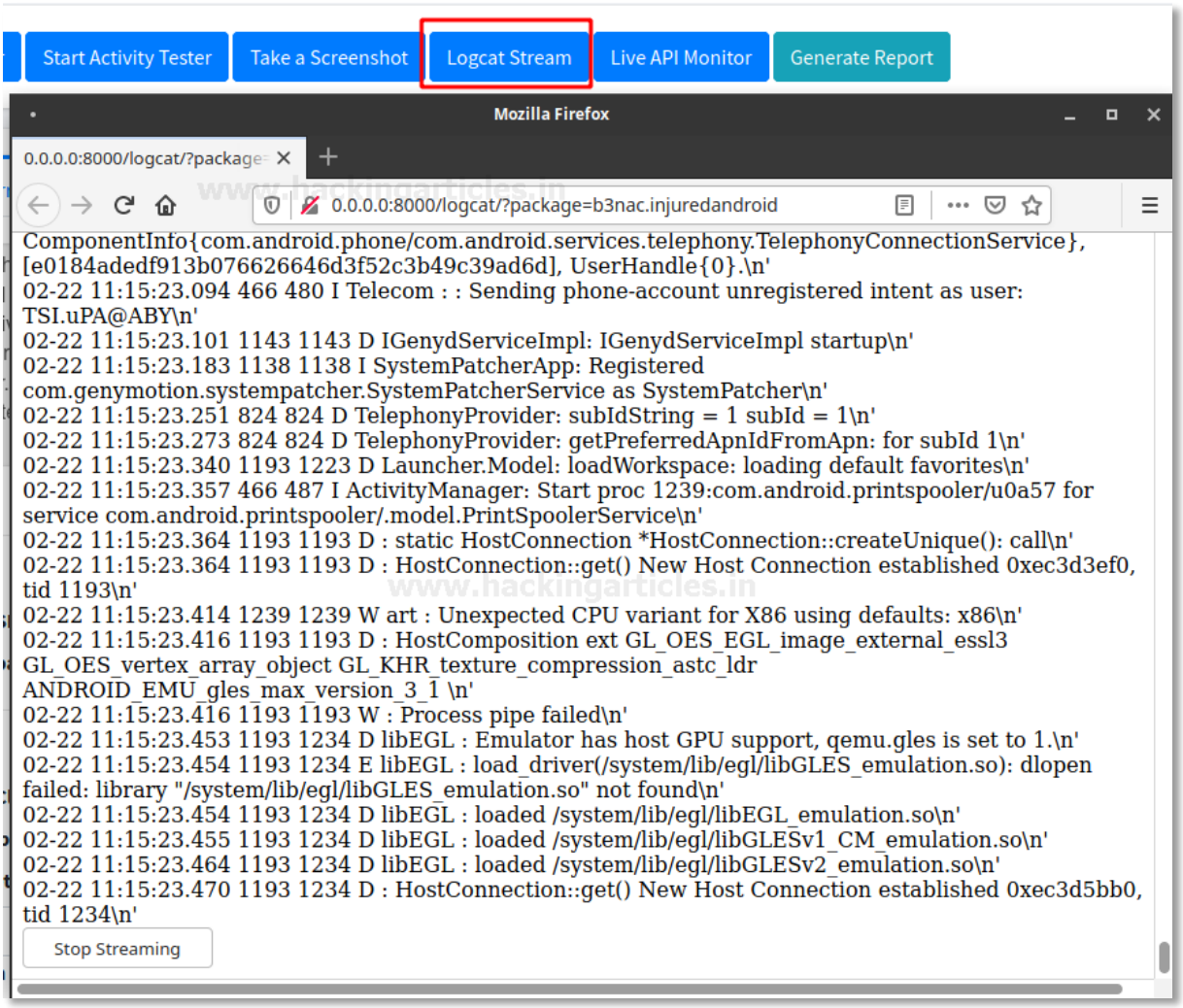


Similarly, if one doesn't have to make do with pre-configured Frida scripts, it is also possible that Frida script be pasted in the text box on the right. There is also a dropdown box that would load those scripts. You can also edit the same.


Frida Code Editor
// https://codeshare.frida.re/@dzonerzy/aesinfo/
Java.perform(function () {
var complete_bytes = new Array();
var index = 0;
var secretKeySpecDef = Java.use('javax.crypto.spec.SecretKeySpec');
var ivParameterSpecDef = Java.use('javax.crypto.spec.IvParameterSpec');
var cipherDef = Java.use('javax.crypto.Cipher');
var cipherDoFinal_1 = cipherDef.doFinal.overload();
var cipherDoFinal_2 = cipherDef.doFinal.overload('[B');
var cipherDoFinal_3 = cipherDef.doFinal.overload('[B', 'int');
var cipherDoFinal_4 = cipherDef.doFinal.overload('[B', 'int', 'int');
var cipherDoFinal_5 = cipherDef.doFinal.overload('[B', 'int', 'int', '[B');
var cipherDoFinal_6 = cipherDef.doFinal.overload('[B', 'int', 'int', '[B', 'int');
var cipherUpdate_1 = cipherDef.update.overload('[B');
var cipherUpdate_2 = cipherDef.update.overload('[B', 'int', 'int');
var cipherUpdate_3 = cipherDef.update.overload('[B', 'int', 'int', '[B');
var cipherUpdate_4 = cipherDef.update.overload('[B', 'int', 'int', '[B', 'int');
var secretKeySpecDef_init_1 = secretKeySpecDef.$init.overload('[B', 'java.lang.String');
var secretKeySpecDef_init_2 = secretKeySpecDef.$init.overload('[B', 'int', 'int',
Available Scripts (Use CTRL to choose multiple)
Load
default
helper
jni_hook_by_address
aes_key

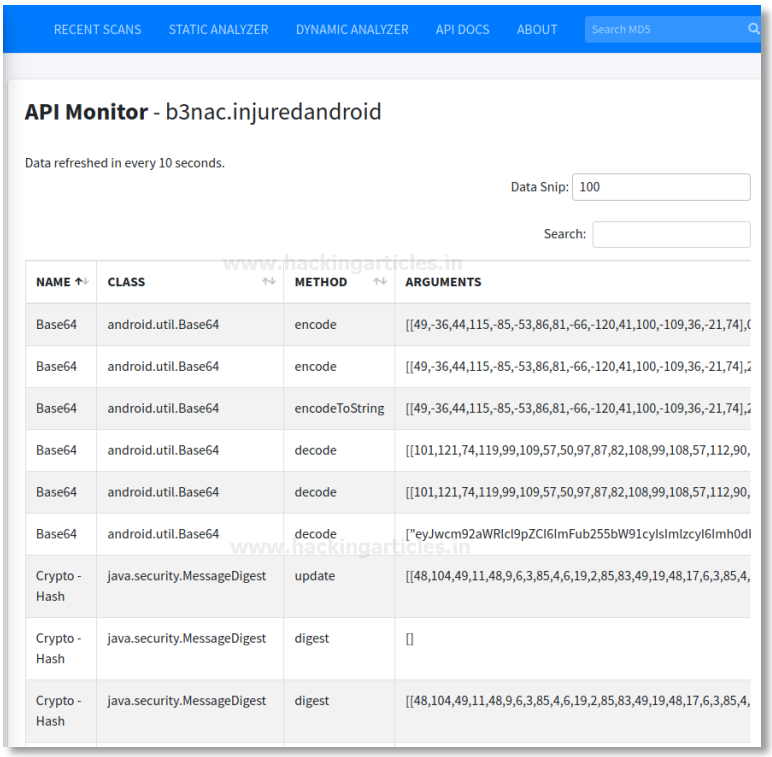
Logcat Stream

Logcat can also be viewed in MobSF's native environment. There's a button at the top menu that lets one view this.

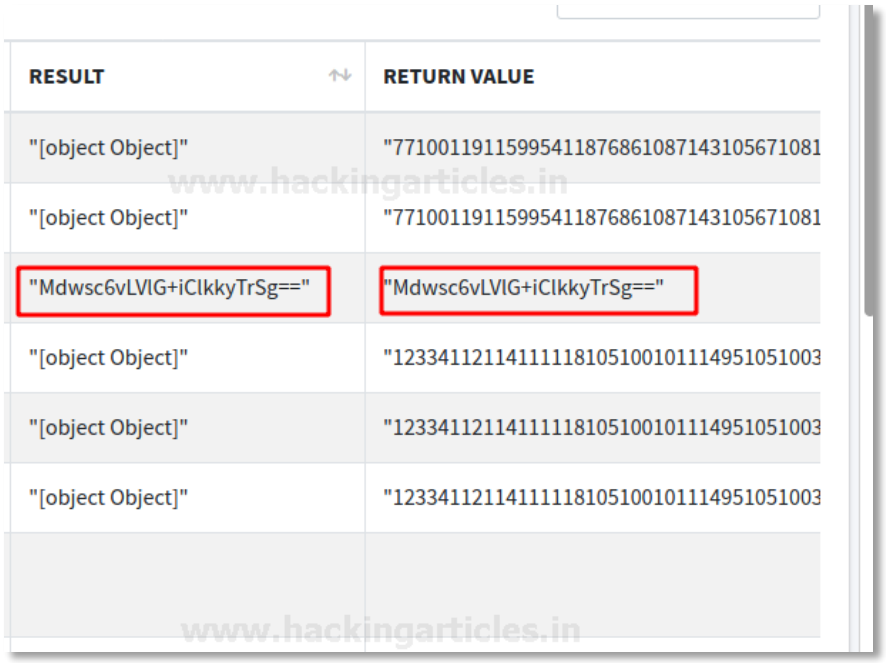


API Monitor

Just like logcat monitors device logs, APIs can also be monitored. APKs use various APIs in real-time to perform various functions, for example, the Base64 library.

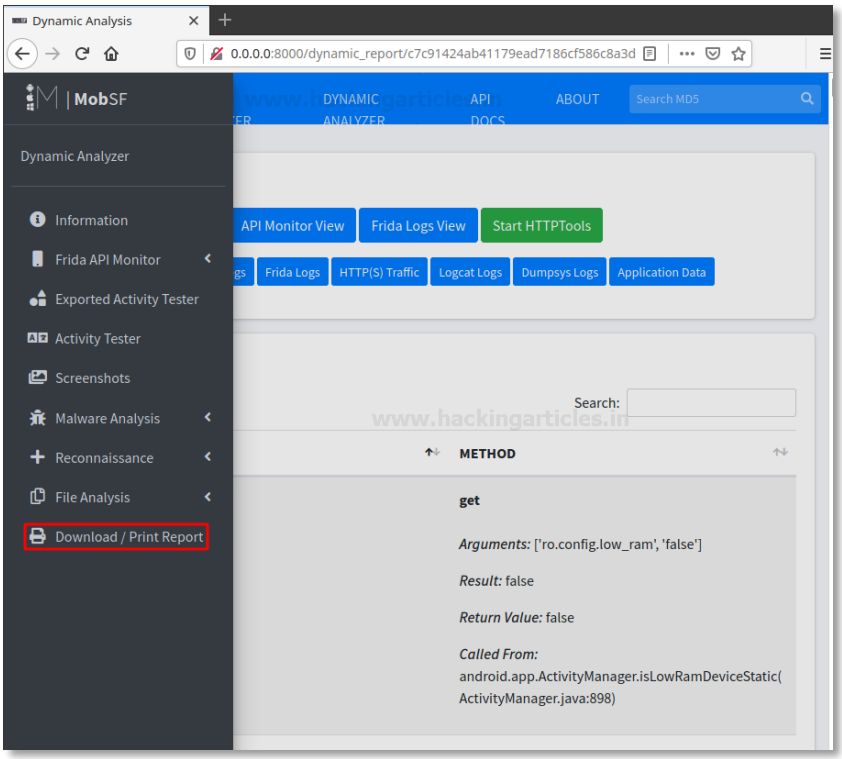


Hence, if a function is using this API and decrypting a value we can see that value here and decode that. For example, down below you can see the return value of once such function in Base64.



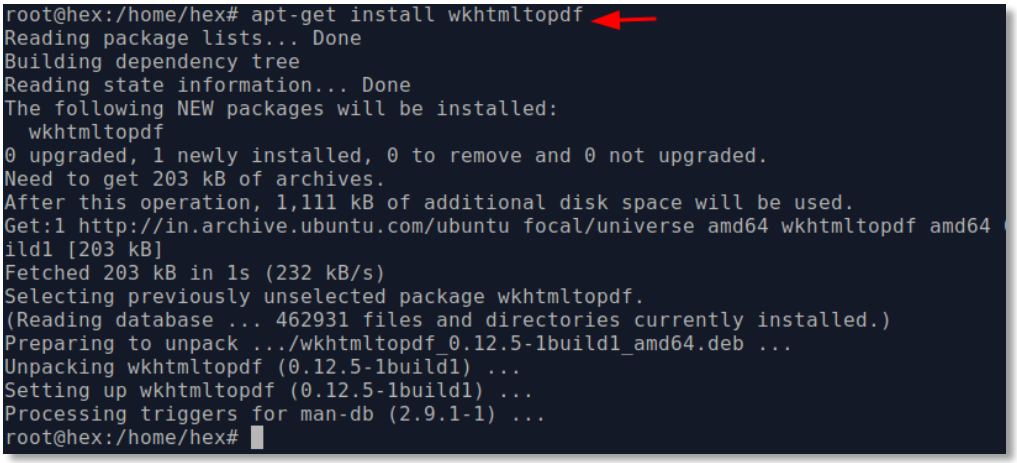
Downloading Reports

Once you have done the analysis, it is possible to download the report by sliding the menu bar slider on the left-hand side and click generate the report.

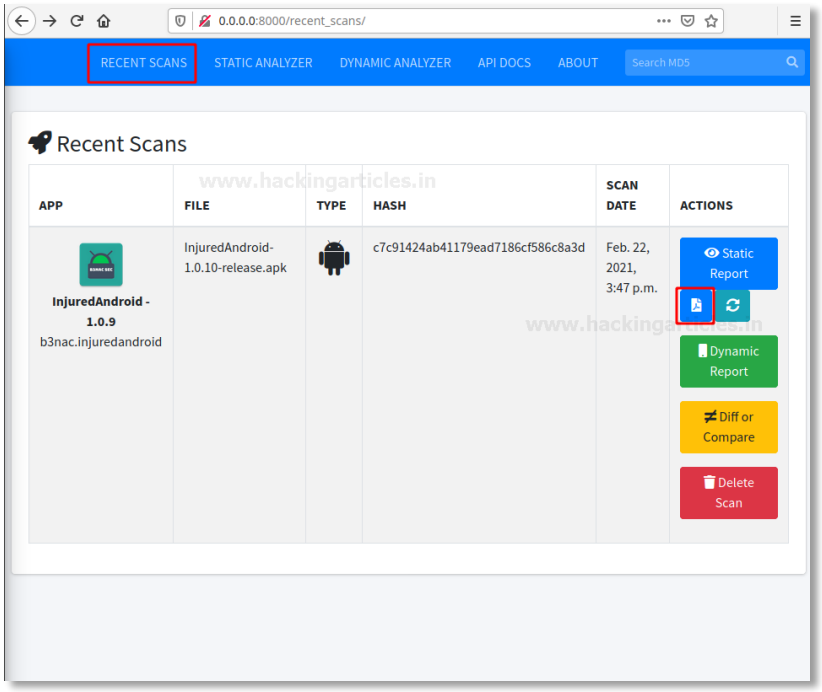


You might notice some errors while generating reports. To resolve this, you can follow the below command and install **wkhtmltopdf** module:

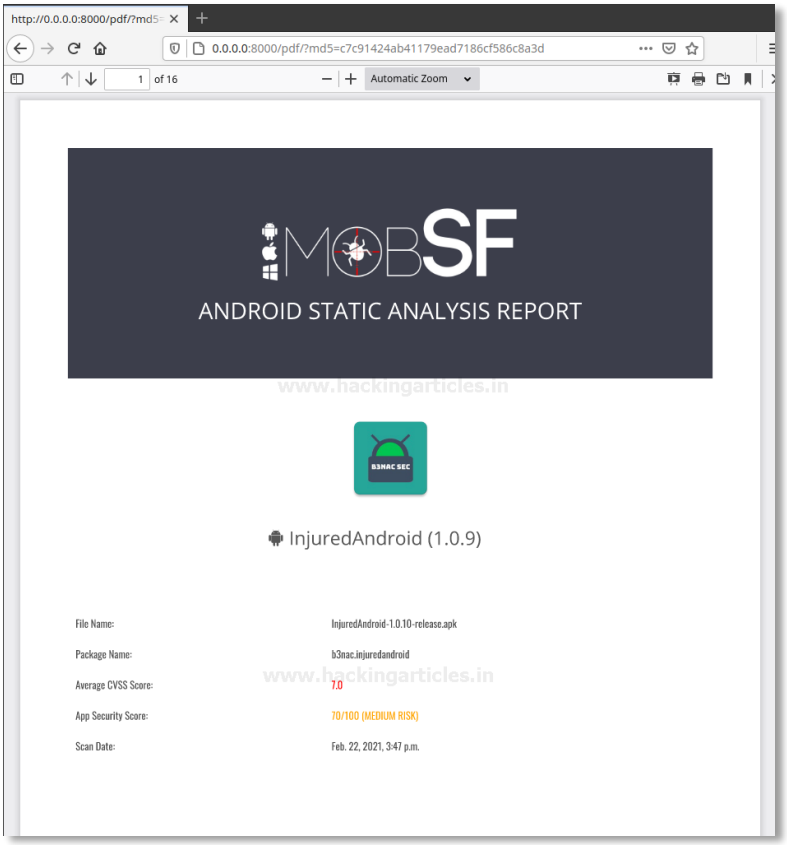
```
apt-get install wkhtmltopdf
```



Now, once again if you click on a recent scan bar, you'll see static and dynamic report generation options.



The report looks something like this:


http://0.0.0.0:8000/pdf/?md5=
0.0.0.0:8000/pdf/?md5=c7c91424ab41179ead7186cf586c8a3d
1 of 16
Automatic Zoom
MOBSF
ANDROID STATIC ANALYSIS REPORT
www.hackingarticles.in
B3NAC SEC
InjuredAndroid (1.0.9)
File Name:
InjuredAndroid-1.0.10-release.apk
Package Name:
b3nac.injuredandroid
Average CVSS Score:
7.0
App Security Score:
70/100 (MEDIUM RISK)
Scan Date:
Feb. 22, 2021, 3:47 p.m.