I J C T A, 9(20), 2016, pp. 19-25
© International Science Press

Survey of Detection Tools For Android Ransomware

Meet Kanwal*, Sanjeev Thakur and Balvinder Singh Saggu*****

ABSTRACT

Android has become a part of every human's life. In android apps, there is a solution for anything and everything that happens in human life. But there is some security issues with android software. There are so many malicious applications developed to infect the smartphones. For detection of these malicious applications, so many detection techniques have been found. Ransomware is a type of malware that infects the user and demands payment through MoneyPak or other payment methods. Report says that the hackers are increasingly targeting mobile users and the number of mobile users affected by ransomware "skyrocketed" during January-March 2016.The most impacted country from mobile ransomware is Germany. Ransomware family Fusob was responsible for 56% of total attacks. In this paper, we discuss about the types of ransomware and detection schemes and tools for android ransomware.

Keywords: Ransomware, FlowDroid, Epicc, Automatic Analysis

1. INTRODUCTION

Ransomware is a type of Malware that infects the personal files of the user and does not permit to access until ransom is paid. This is a malicious software, there are various types of ransomware found till 2016. Basically it is of two types:-

- Lock screen ransomware (WinLocker Ransomware)
- Crypto ransomware (File Encryptor Ransomware)

In lock screen ransomware, it does not encrypt the personal files, it just lock the screen and demands payment. And Crypto ransomware encrypts the personal files/folders. In this type of ransomware, files are encrypted and after encryption, user is informed that his data is encrypted and will not be decrypted until ransom is paid. Analysis shows that Malware uses AES+RSA Encryption.

Though RSA uses asymmetric keys; one is public which is accessible by outside party and the other is private key, only kept by the user. While AES is a symmetric key cryptography, which has only one key i.e one key uses for both encryption and decryption. AES key is used for file encryption. Encrypted files are used for storing AES key for decryption. An RSA public key is encrypted with this AES key either we can say , for decryption there is a need of a private key. This ransomware is divided into three sub types:-

- Private key cryptosystem Ransomware (PrCR)
- Public key cryptosystem Ransomware (PuCR)
- Hybrid cryptosystem Ransomware (HCR)

* Department of Computer Science Amity University, Noida, Uttar Pradesh, India, *Email: meetkanwalsodhi@gmail.com*

** Department of Computer Science Amity University, Noida, Uttar Pradesh, India, *Email: sthakur3@amity.edu*

*** Department of Computer Science Shriram Institute of Management & Technology, Kashipur, uttarakhand, India, *Email: meetkanwalsodhi@gmail.com*

In PrCR, the view of the ransomware writer and the view of the malware analyst are symmetric. For making the view asymmetric, the key must be removed from the malware analyst's view, but it is possible to recover the key again by reverse engineering or sometimes with brute forcing. But the fact that everything is visible to the analyst, is the major disadvantage of using this cryptosystem.

In PuCR, there is a pair of keys known as Public key and Private key or we can say encryption key and decryption key respectively. Public key is used for encrypting data on the victim machine, while private key is kept by the malware writer secretly.

So, it would not be possible for the malware analyst to detect this private key and this pair of key is generated only once, so the data is decrypted only when victim is agreed to pay the ransom in exchange of the private key.

But this approach also has so many drawbacks ,in this, malware writer cannot free one victim at a time, he has to hold everyone until all victim users pay their ransom because if he frees one victim, that victim could reveal the private key, it can be overcomed if PuCR generates multiple key pairs. Another drawback is that the symmetric encryption schemes are much faster than asymmetric encryption schemes.

To overcome above mentioned drawbacks, HCR is developed. In this case, a pair of asymmetric keys are generated again and the public key is place in malware payload. But for the data encryption process a random secret key is generated on each victim machine, and the captive data are encrypted using this key and a fast symmetric cipher. The random generated secret key is encrypted using the public key and only stored in this way. In this case the adversary is not required to disclose his private key. The malware writer demands the ransom and for decryption, the cipher text of the random secret key is sufficient.

He then decrypts the secret key using the private key and sends it back to the victim. In this method ,with a high probability each victim has a unique key, and so publishing of the decryption key is of no help to other victims[5].

There are many file encrypting ransomwares, such as:-

- SimpLocker
- CryptoLocker
- CTB-Locker
- Torrent Locker

Android Defender, the first actual ransomware ,which was found in mid 2013,is a kind of counterfeit antivirus ,which is appeared to the victims that they are dealing with a legitimate security application ,everything seems like it is done for phone's security issues but actually it was a fake anti virus and asked to pay the ransom.

There is a ransomware named Police ransomware which come under the category of Lockscreen Ransomware.It uses another shuddery tactic, according to which ,they threatened victims by displaying a message from a Law Enforcement Agency such as FBI, claiming that outlawed activities has been traced on their device. Reveton is the example of this family.

Previously we uninstalled the malicious application in safe mode, but then in 2015,Lockerpin was discovered. If user has MDM solution on their device or the device was previously rooted, then there is a possibility of resetting the PIN if the user is infected by this kind of ransomware otherwise, factory reset is the last option to get rid of this virus.

2. SURVEY OF DETECTION TOOLS AND SCHEMES

There are many techniques to detect malicious apps, for those people who do not have much knowledge about Android, there is a detection technique known as automatic analysis which includes two types of analysis:

- Static analysis
- Dynamic analysis

In static analysis, malicious code and security threats are detected in a non-runtime environment. But we cannot detect all the threats at compile time, and that is why we need dynamic analysis. In dynamic analysis threats and malicious code is detected at runtime or we can say when the code is executed we get the result if code is malicious or not. But some researchers found that we must use both the techniques to detect the threat, and therefore automatic analysis concept was proposed, which combines both static analysis and dynamic analysis for inspecting threats and malicious apps. The key of this concept is the unification of data states and software execution on the critical path[1]. In this two-phase approach first part is static analysis and second part is dynamic analysis.

2.1. Static Analysis

In static analysis, original file of any app known as APK file undergoes de-compilation and is repackaged by some unknown third party where some malicious resources are injected into it afterwards this repackaged code is decoded again. Feature extraction and feature comparison is done but all these processes are done in a non- execution time. So we are not able to detect all the security threats.

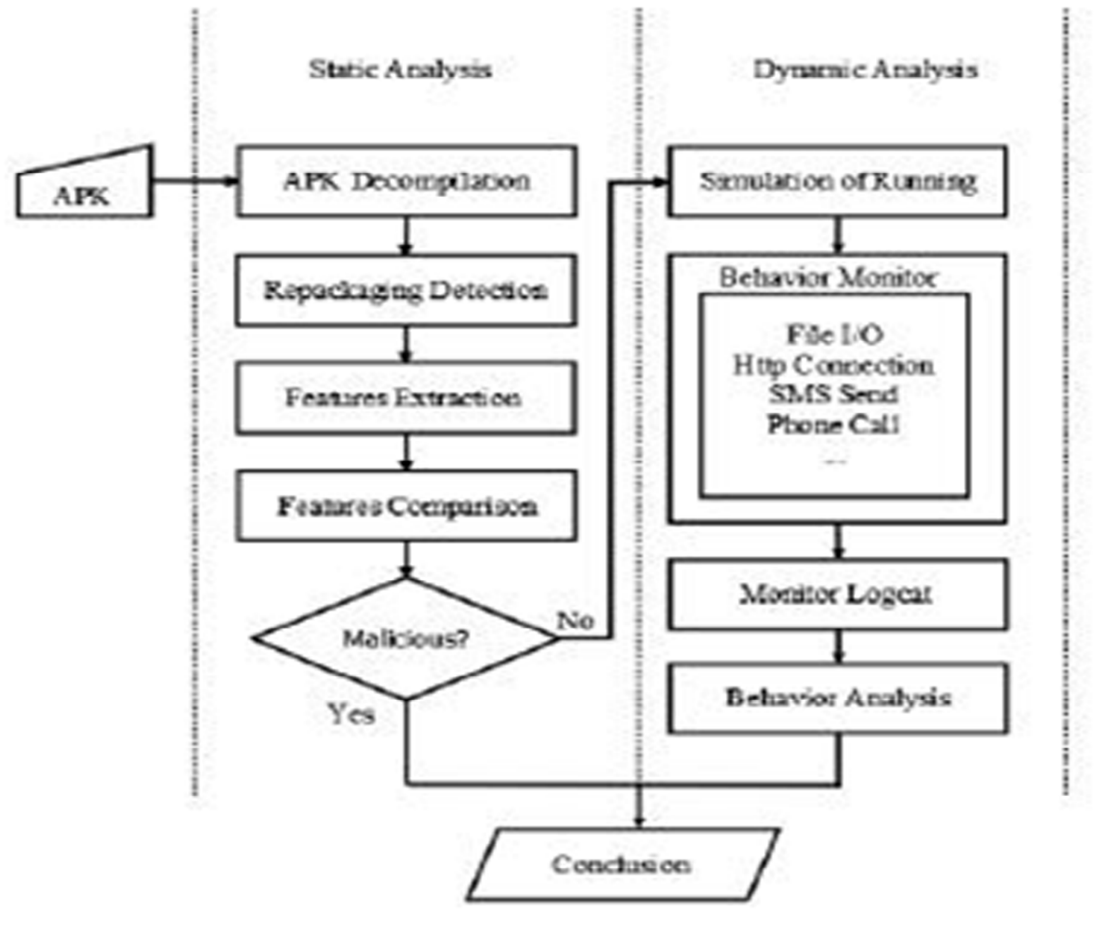


Figure 1: Automatic Analysis Overview[1]

2.2. Dynamic Analysis

The In dynamic analysis we check and control the behavior of the executing application at run time. The behavior analysis includes 4 behaviors:

- Critical path and data flow
- Malicious domain access
- Malicious charges
- Bypassing the Android permission

In real-time analysis, an app known as Taint analysis is used to detect malicious content. This app is used, when only permission system cannot prevent sensitive data flow from being leaked. Taint analysis detects whether the sensitive information is safe or not and to check if any malicious data has infected the Secure data. It has two phases:

The first phase includes FlowDroid and Epicc analysis. The FlowDroid Static analysis is context flow object, field sensitive and Android app life cycle aware[3]. FlowDroid identifies sources(including intents received),flow of the data within each component, and sinks(including intents sent).And EPICC identifies characteristics of intents sent by a component[2]. In phase I, each app is analyzed individually where received intents are termed as sources and the send intents are termed as sinks. The output of phase 1 includes:

- Flows within each component found by FlowDroid.
- Identification of the properties of sent intents as found by Epicc.
- Intent filters of each component, extracted from manifest.

An intent ID is assigned to every source code location that sends an intent. Sent intents with distinct IDS are considered distinct sinks while intents with the same ID are combined together[2]. Phase-2 analysis is done on the output of phase-1 analysis. Analysis is done on a particular set of apps. It provides knowledge about data flows from a received intent to a sent intent.

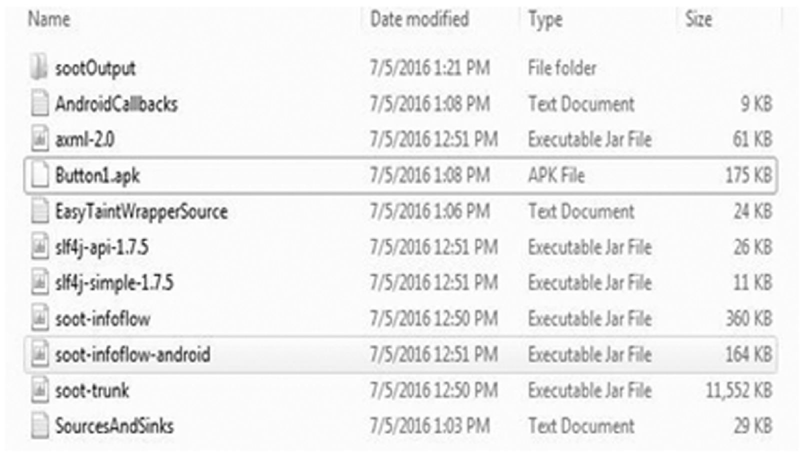
3. EXPERIMENT RESULTS AND ANALYSIS

An experiment was conducted using FlowDroid nightly builds as provided on GitHub[6]. The nightly builds requires you to download five jar files namely:

- Soot bundle
- soot-infoflow
- soot-infoflow-android
- Slfj4-api-1.7.5
- slf4j-simple-1.7.5
- axml-2.0

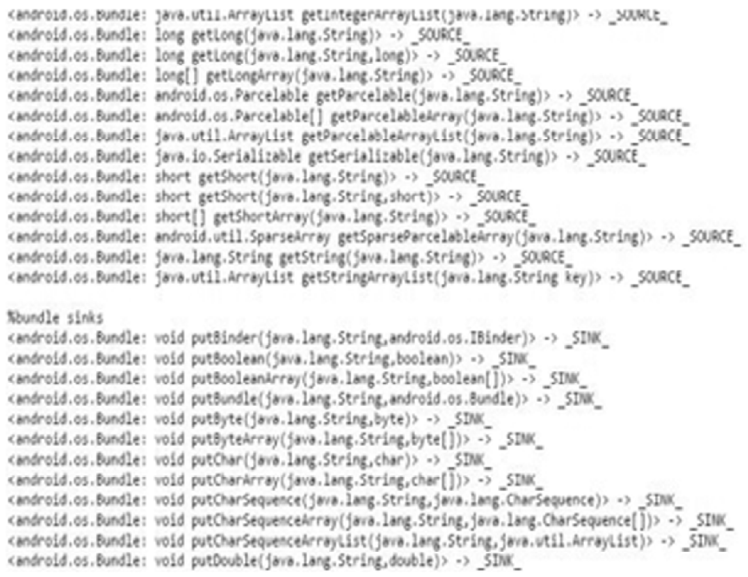
All these files are to saved in a same directory. After this we had to configure FlowDroid with the sources and sinks that we want to use. For this we used a fairly comprehensive list provided by them.. Two other required configuration files were downloaded for the Taint Wrappers and for the Callbacks. After downloading all the files the experiment was run by using the following command on the cmd.

java-Xmx4g-cpsoot-trunk.jar;soot-infoflow.jar;soot-infoflow-android.jar;slf4j-api-1.7.5.jar;slf4j-simple-7.5.jar;axml-2.0.jarsoot.jimple.infoflow.android.TestApps.Test"D:\Callbacks_Button1.apk" D:\Tools\AndroidSDK\sdk\platforms [6].



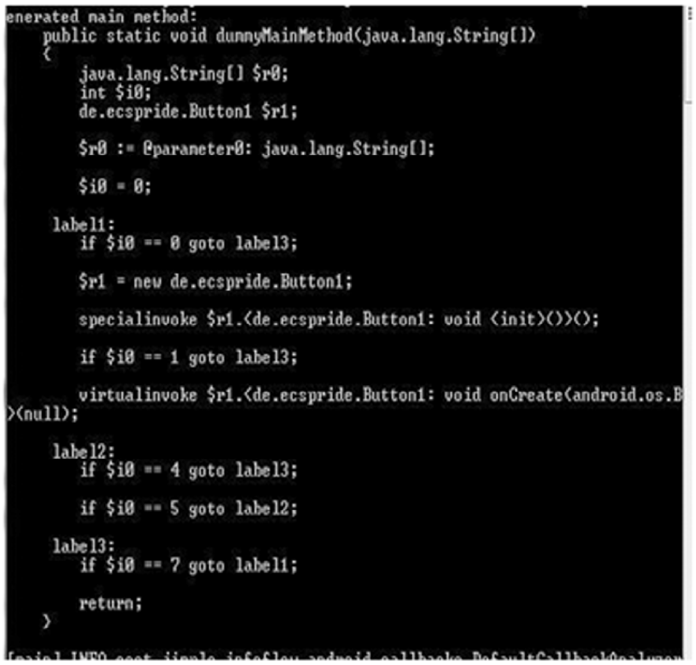
Name	Date modified	Type	Size
sootOutput	7/5/2016 1:21 PM	File folder	
AndroidCallbacks	7/5/2016 1:08 PM	Text Document	9 KB
axml-2.0	7/5/2016 12:51 PM	Executable Jar File	61 KB
Button1.apk	7/5/2016 1:08 PM	APK File	175 KB
EasyTaintWrapperSource	7/5/2016 1:06 PM	Text Document	24 KB
slf4j-api-1.7.5	7/5/2016 12:51 PM	Executable Jar File	26 KB
slf4j-simple-1.7.5	7/5/2016 12:51 PM	Executable Jar File	11 KB
soot-infoflow	7/5/2016 12:50 PM	Executable Jar File	360 KB
soot-infoflow-android	7/5/2016 12:51 PM	Executable Jar File	164 KB
soot-trunk	7/5/2016 12:50 PM	Executable Jar File	11,552 KB
SourcesAndSinks	7/5/2016 1:03 PM	Text Document	29 KB

(a)



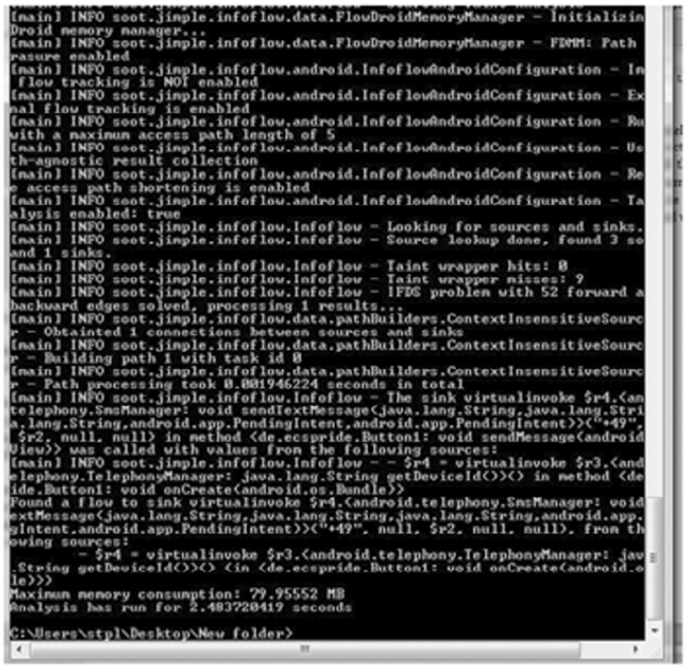
```
<android.os.Bundle: java.util.ArrayList getIntegerArrayList(java.lang.String)> -> _SOURCE_
<android.os.Bundle: long getLong(java.lang.String)> -> _SOURCE_
<android.os.Bundle: long getLong(java.lang.String,long)> -> _SOURCE_
<android.os.Bundle: long[] getLongArray(java.lang.String)> -> _SOURCE_
<android.os.Bundle: android.os.Parcelable getParcelable(java.lang.String)> -> _SOURCE_
<android.os.Bundle: android.os.Parcelable[] getParcelableArray(java.lang.String)> -> _SOURCE_
<android.os.Bundle: java.util.ArrayList getParcelableArrayList(java.lang.String)> -> _SOURCE_
<android.os.Bundle: java.io.Serializable getSerializable(java.lang.String)> -> _SOURCE_
<android.os.Bundle: short getShort(java.lang.String)> -> _SOURCE_
<android.os.Bundle: short getShort(java.lang.String,short)> -> _SOURCE_
<android.os.Bundle: short[] getShortArray(java.lang.String)> -> _SOURCE_
<android.os.Bundle: android.util.SparseArray getSparseParcelableArray(java.lang.String)> -> _SOURCE_
<android.os.Bundle: java.lang.String getString(java.lang.String)> -> _SOURCE_
<android.os.Bundle: java.util.ArrayList getStringArrayList(java.lang.String key)> -> _SOURCE_

%bundle sinks
<android.os.Bundle: void putBinder(java.lang.String,android.os.IBinder)> -> _SINK_
<android.os.Bundle: void putBoolean(java.lang.String,boolean)> -> _SINK_
<android.os.Bundle: void putBooleanArray(java.lang.String,boolean[])> -> _SINK_
<android.os.Bundle: void putBundle(java.lang.String,android.os.Bundle)> -> _SINK_
<android.os.Bundle: void putByte(java.lang.String,byte)> -> _SINK_
<android.os.Bundle: void putByteArray(java.lang.String,byte[])> -> _SINK_
<android.os.Bundle: void putChar(java.lang.String,char)> -> _SINK_
<android.os.Bundle: void putCharArray(java.lang.String,char[])> -> _SINK_
<android.os.Bundle: void putCharSequence(java.lang.String,java.lang.CharSequence)> -> _SINK_
<android.os.Bundle: void putCharSequenceArray(java.lang.String,java.lang.CharSequence[])> -> _SINK_
<android.os.Bundle: void putCharSequenceArrayList(java.lang.String,java.util.ArrayList)> -> _SINK_
<android.os.Bundle: void putDouble(java.lang.String,double)> -> _SINK_
```

(b)



```
enerated main method:
    public static void dummyMainMethod(java.lang.String[])
    {
        java.lang.String[] $r0;
        int $i0;
        de.ecspride.Button1 $r1;

        $r0 := @parameter0: java.lang.String[];

        $i0 = 0;

     label1:
        if $i0 == 0 goto label3;

        $r1 = new de.ecspride.Button1;

        specialinvoke $r1.<de.ecspride.Button1: void <init>()>();

        if $i0 == 1 goto label3;

        virtualinvoke $r1.<de.ecspride.Button1: void onCreate(android.os.B
>(null);

     label2:
        if $i0 == 4 goto label3;

        if $i0 == 5 goto label2;

     label3:
        if $i0 == 7 goto label1;

        return;
    }
```

(c)



```
[main] INFO soot.jimple.infoflow.data.FlowDroidMemoryManager - Initializin
Droid memory manager...
[main] INFO soot.jimple.infoflow.data.FlowDroidMemoryManager - FDMM: Path
rasure enabled
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - In
 flow tracking is NOT enabled
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - Ex
nal flow tracking is enabled
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - Ru
with a maximum access path length of 5
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - Us
th-agnostic result collection
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - Re
e access path shortening is enabled
[main] INFO soot.jimple.infoflow.android.InfoflowAndroidConfiguration - Ta
alysis enabled: true
[main] INFO soot.jimple.infoflow.Infoflow - Looking for sources and sinks.
[main] INFO soot.jimple.infoflow.Infoflow - Source lookup done, found 3 so
and 1 sinks.
[main] INFO soot.jimple.infoflow.Infoflow - Taint wrapper hits: 0
[main] INFO soot.jimple.infoflow.Infoflow - Taint wrapper misses: 9
[main] INFO soot.jimple.infoflow.Infoflow - IFDS problem with 52 forward a
backward edges solved, processing 1 results...
[main] INFO soot.jimple.infoflow.data.pathBuilders.ContextInsensitiveSourc
r - Obtainted 1 connections between sources and sinks
[main] INFO soot.jimple.infoflow.data.pathBuilders.ContextInsensitiveSourc
r - Building path 1 with task id 0
[main] INFO soot.jimple.infoflow.data.pathBuilders.ContextInsensitiveSourc
r - Path processing took 0.001946224 seconds in total
[main] INFO soot.jimple.infoflow.Infoflow - The sink virtualinvoke $r4.<an
telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.Stri
a.lang.String,android.app.PendingIntent,android.app.PendingIntent)>("+49",
 $r2, null, null)> in method <de.ecspride.Button1: void sendMessage(android
View)> was called with values from the following sources:
[main] INFO soot.jimple.infoflow.Infoflow - - $r4 = virtualinvoke $r3.<and
elephony.TelephonyManager: java.lang.String getDeviceId()>() in method <de
ide.Button1: void onCreate(android.os.Bundle)>
Found a flow to sink virtualinvoke $r4.<android.telephony.SmsManager: void
extMessage(java.lang.String,java.lang.String,java.lang.String,android.app.
gIntent,android.app.PendingIntent)>("+49", null, $r2, null, null), from th
owing sources:
        - $r4 = virtualinvoke $r3.<android.telephony.TelephonyManager: jav
.String getDeviceId()>() (in <de.ecspride.Button1: void onCreate(android.o
le)>)
Maximum memory consumption: 79.95552 MB
Analysis has run for 2.483720419 seconds

C:\Users\stpl\Desktop\New folder>
```

(d)

Figure 2: (a) Files download to perform FlowDroid experiment (b) List of sources and sinks provided to the file (c) Dummy method created by the process (d) Final result of the experiment

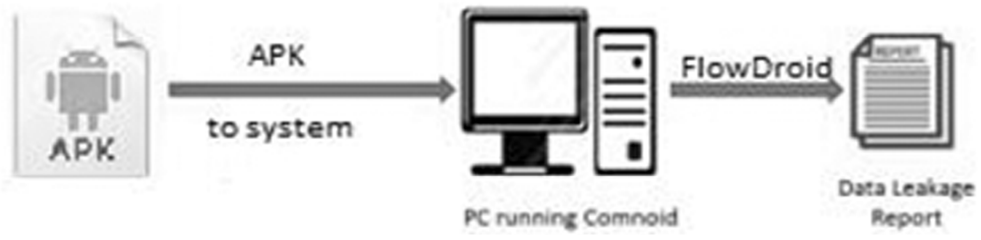


Figure 3: Flowchart of the experiment performed

The command involves the names of the executable jar files and the location of the apk file we want to perform the test on. The command also contains the location of platforms directory inside the Android SDK which we had download from Google. It was mandatory to use Google's official version, not the one available on GitHub which were too large to be practically usable with FlowDroid. In the example command-line above, we configure the Java VM to use a Maximum heap size of 4 GB. For some large applications, this might not be enough. In general, the more memory we can allocate, the more precise options we can use and the bigger applications we can analyze.

The experiment started with loading all the jars, and after binding to all of them, it started the actual process in which it made various dummy methods and executed them to find flows and sinks. After running for a few seconds it came up with a result in which it had identified a flow to sink virtual invoke. Along with the result the method in the end also stated the time and the memory used by the process to be completed. The results from the above run experiment contains the data leakage reports. The FlowDroid successfully found a potential data leakage threat.

4. CONCLUSION

In this paper, we presented several detection schemes and tools to detect ransomware. These techniques include Automatic Analysis: static analysis and dynamic analysis, Taint Analysis: FlowDroid and Epicc and the tool DidFail which is used for implementing the taint analysis. FlowDroid is used for static taint analysis. Epicc performs inter component communication in an android app set. DidFail performs in two phases.Phase-2 takes the output of phase-1 to get result. We performed an experiment as per the steps mentioned by FlowDroid Nightly Builds on Github.

REFERENCES

[1] Tianda Yang,Yu Yang,Kai Qian,Dan Chia-Tien Lo.Automated Detection and Analysis for Android Ransomware.In IEEE,pages 1338-1343,2015.

[2] William Klieber,Lori Flynn,Amar Bhosale,Limin Jia,Lujo Baner.Android Taint Flow Analysis for App Sets. In IEEE, pages 1-6,2014.

[3] C. Fritz, S. Arzt, S. Rasthofer, E. Bodden, A. Bartel, J. Klein, Y. le Traon, D. Octeau, and P. McDaniel. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In Proc. PLDI, 2014.

[4] Jonathan Burket,Lori Flynn,William Klieber,Jonathan Lim,Wei Shen,William Snavely.Making DidFail Succeed: Enhancing the CERT Static Analyzer for Android App Sets.March 2015.

[5] Connection-Monitor & Connection-Breaker: A Novel Approach for Prevention and Detection of High Survivable Ransomwares.

[6] https://github.com/secure-software-engineering/soot-infoflow-android/wiki