AndroidAnti-Malware Analysis

Mr.Sharad R.Kurhade
ASM INSTITUTE OF MANAGEMENT & COMPUTER STUDIES (IMCOST), THANE, MUMBAI
University of Mumbai

Mr.Nayan D. Gite
ASM INSTITUTE OF MANAGEMENT & COMPUTER STUDIES (IMCOST), THANE, MUMBAI
University of Mumbai

***Abstract*—Security is one of the main concerns for Android users today. Users will access the Internet intensively and everyone can develop applications for Android. Android is currently the most popular smart phone operating system. Antivirus software promises to effectively protect against malware on mobile devices and many products are available for free or at reasonable prices. Their effectiveness is supported by various reports, attesting very high detection rates. However, a more detailed investigation is required in order to understand the real risk level arising from malware for the Android platform. Neither do the exceedingly high numbers of different malware variants reflect the real threat in comparison to other.**
One key feature of such devices is their ability to incorporate third-party apps from a variety of markets. This poses strong security and privacy issues to users and infrastructure operators, particularly through software of malicious (or dubious) nature that can easily get access to the services provided by the device and collect sensory data and personal information. Malware in current Android devices –mostly smart phones and tablets– have rocketed in the last few years, in some cases supported by sophisticated techniques purposely designed to overcome security architectures currently in use by such devices. Even though important advances have been made on malware detection in traditional adapting those techniques to smart devices is a challenging problem.

Keywords— Android,Security, Anti-virus, Exploits, Malware, Spyware,Security Assessment, Malware transformation.

I.INTRODUCTION

As android is an open source operating system, so developers can upload their application without any certificate check whereas they can upload self-signed applications which can be done without any help or assistance.

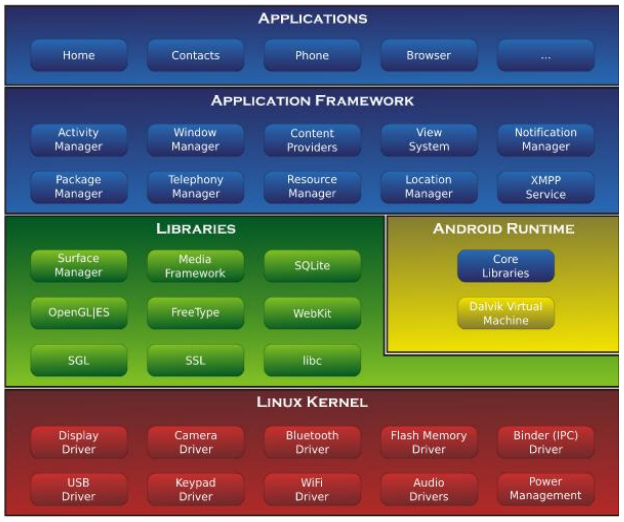


Fig. 1 Android Architecture

Android OS consist of various layers such as application, application framework, libraries, and Linux kernel. All these layers are vulnerable to various types of attacks, of which application layer is the most vulnerable because an average user mostly interacts with this layer to perform some basic functionality such as making phone call, accessing internet through web browser etc. For example, recent vulnerability allows a malicious app to bypass intended access restriction and remove the device lock activated by user. Malware developer exploits these vulnerabilities to get into the system. However these vulnerabilities can be fixed by regular updates by Android OS through patches.

II. **Mobile malware**

Malware is portmanteau term for malicious software and is software program whose aim is to do malicious works on mobile device like stealing information, spying on user activity or to disable the mobile system. The most common types of malicious programs on mobile platforms are Trojan horse, worm and spyware.

Trojan horse: It requires user interaction to be activated. Once activated it can do malicious activities such as disabling phone, deactivating other application or send certain information to remote server.

Worm:worm reproduces itself in a device which it infects. Worm can also be send through SMS or MMS and do not require any user interaction. They did not do any malicious activities but can slow down the infected device.

Spyware: spyware spy on user activities such as browsing, location information, password, credit card numbers and other such sensitive information. Adware which is a kind of spyware use location information to send advertisement to that device.

III. **Anti-malware**

Anti-malware is tools and program to remove malware from mobile device. Whenever a new application installed from Internet, Android OS broadcast the message of installation of new application. Anti-malware immediately scans the new application and look for any thread, if found it alert the user and do appropriate actions to avert those thread. Other than internet, malware can get into mobile device by sharing .apk file from other mobile device through wifi or Bluetooth. Thus anti-malware application make sure to scan any new application from whichever sources they get into the mobile device.
To install malware on computers, mobiles phones criminals exploit any vulnerabilities that they can find such as human vulnerabilities as well as technical ones like weak security policies. Increasingly, the easiest route into a system is *social engineering*: luring users (through websites, downloads, and online advertising) to navigate to sites that install malware on their Devices
Malware can be spread through such vectors as downloaded email and IM attachments, applications shared on social networking sites, files shared peer-to-peer or on network shares.

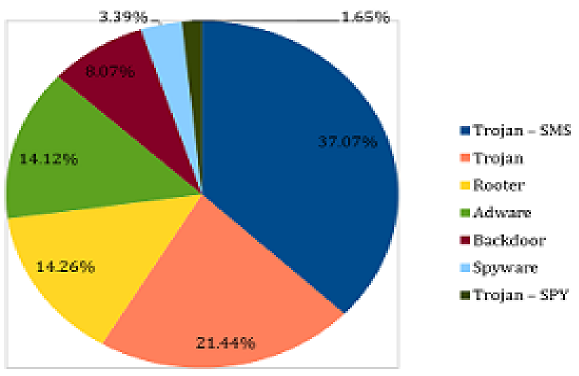


Fig. 2 Different Types of Attack on Android OS

IV. **Malware Detection techniques**

Android is an operating system for mobile devices such smart phones and tablets. It is based on the Linux kernel and provides a middleware implementing subsystems such as telephony, window management, management of communication with and between applications, managing application lifecycle with the proliferation of malware, there are now scores of both free and paid anti-malware products available in the official Android market. Many are from obscure developers while well-established, mainstream antivirus vendors offer others. In order to get an insight on the workings of the anti-malware products, we briefly describe the necessary parts of the Android security model. Android achieves application sandboxing by means of Linux UIDs. Every application (with a few exceptions relating to how applications are signed) is given a separate UID and most of the application resources remain hidden from other UIDs. Android anti-malware products are treated as ordinary third party applications and have no additional privileges over other applications. This is in contrast with the situation on traditional platforms such as Windows and Linux where antivirus applications run with administrator privileges. An important implication of this is that these anti-malware tools are mostly incapable of behavioural monitoring and do not have access to the private files of the application. The original application packages however remain intact and are readable by all applications. (Copy protected application packages are no tread able by all applications but this feature is deprecated; paid applications are reportedly kept encrypted since Android 4.1.Note that malware have not been found in paid apps.) These application packages may thus be used for static, signature based malware detection. Moreover, Android provides a broad cast when a new application is installed. All the anti-malware applications we study have the ability to scan applications automatically immediately following their installation, most likely by listening to this broadcast. Android also provides a Package Manager API, which allows applications to retrieve all the installed packages. The API also allows getting the signing keys of these packages
And the information stored in their Android Manifest such as the package name, names of the components declared, permissions declared and requested, and so on. Anti-malware applications have the opportunity to use information from this API as well for malware detection.

A. Malware Detection Using Signatures

Signature-based detection uses a set of patterns that act like fingerprints to uniquely identify malicious programs. Antivirus vendors are constantly adding to their signature databases to keep up with emerging viruses, worms, and related malware. Once a signature has been identified for a malicious program, the code can be readily blocked at the network or device levels. While developing malware transformations, It is important to consider what kind of signatures anti malware tools may use against malware. Signatures have traditionally been in the form of fixed strings and regular expressions. Anti-malware tools may also use chunks of code, an instruction sequence or API call sequence as signatures. Signatures that are more sophisticated require a deeper static analysis of the given sample. The fundamental techniques of such an analysis comprised at and control flow analysis. Analysis may be restricted with in function boundaries (intra-procedural analysis) or may expand to cover multiple functions (inter-procedural analysis).

B. Semantics-based Malware Detection

In order to evade detection, malware writers (hackers) frequently use obfuscation to morph malware. Malware detectors that use a pattern-matching approach (such as commercial virus scanners) are susceptible to obfuscations used by hackers. The fundamental deficiency in the pattern-matching approach (signature based) to malware detection is that it is purely syntactic and ignores the semantics of instructions.

C. Cloud-based detection

Because of limited computational power and energy sources, smartphones don't carry fully featured security mechanisms. Running a simple file scanner on an Android HTC G1 device takes nearly 30 minutes and reduces the battery life by 2 percent.[11] A scanning application reportedly runs 11.8 times slower on an HTC G1 than on a desktop PC, highlighting the need for new mobile malware analysis techniques.[12]

Figure 4a shows Paranoid Android (PA), a cloud-based malware protection technique that moves security analysis and computations to a remote server that hosts multiple replicas of mobile phones running on emulators.[11] A tracer, located in the smartphone, records all the necessary information required to replay the mobile application's execution. The tracer transmits the recorded information to the cloud-based replayer, which replays the execution in the emulator. The replayer can deploy several security checks, such as dynamic malware analysis, memory scanners, system call anomaly detection, and commercial antivirus scanning, from the cloud's ample resources.

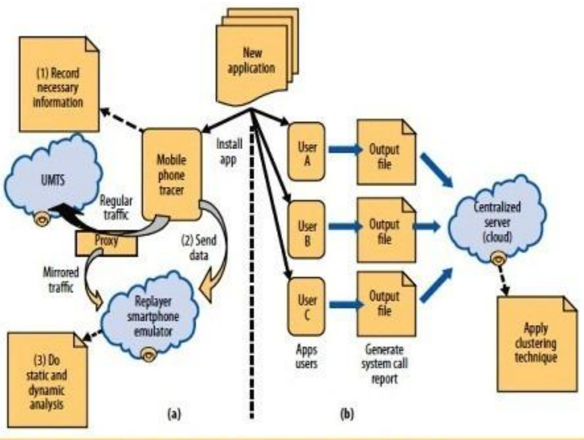

(1) Record necessary information
New application
Mobile phone tracer
Install app
UMTS
Regular traffic
Proxy
(2) Send data
Mirrored traffic
Replayer smartphone emulator
(3) Do static and dynamic analysis
(a)
User A
User B
User C
Output file
Output file
Output file
Centralized server (cloud)
Apply clustering technique
Apps users
Generate system call report
(b)

Figure 4. Cloud-based malware protection techniques: (a) Paranoid Android and (b) Crowdroid.

PA uses a proxy to temporarily store inbound network traffic information so that the phone can save energy by not sending this data back to the server. Instead, the server can directly contact the proxy to get the network traffic information needed to successfully replay the execution. However, PA incurs some significant overhead, increases the CPU load by 15 percent, and consumes 30 percent more energy during heavyweight tasks. Furthermore, because tracing systems implement the tracer module in the user space, the calls incur heavy overhead compared to native execution. For example, tracing a single system call such as read() takes 0.7 milliseconds in the user space, but less than 0.1 millisecond in the kernel.

Figure 4b depicts Crowdroid, a behaviour-based mobile malware detection technique for Android.[13] Crowdroid is a lightweight client application that monitors system calls invoked by the target mobile application, preprocesses the calls, and sends them to the cloud, where a clustering technique helps determine whether the application is benign or malicious. Increased use of Crowdroid will result in improved malware detection, but using the approach initially might cause from false positives as the sample size is still very small. Moreover, it isn't clear how users will react when they're asked to send application behaviour to a third party, and total dependence on user behaviour might not produce accurate results.

CloudAV is a cloud-based antivirus file scanning mechanism, but it lacks the features required to detect zero-days attacks, remote exploits, and memory-resident attacks.

V. Malware transformation

Malware can be transformed into different variants keeping the same behaviour. Thus malware programmers often use transformation techniques to get undetected by various anti-malware software's. Malware transformations are categorized into two classes repackaging and code obfuscation. Both transformation process takes .apk file as an input and generate different variants of the same .apk file. However the output .apk file preserves the logic and functionality of the input .apk file.

A. Repackaging:

Repackaging method generate different .apk file keeping the input .apk file intact. This makes malware easily deployable and the functionality of the malware is also preserved. Following are some different types of repackaging methods used to transform the malware into different variants.

Alignment: In this method the data in .apk files are realigned so that different content can be generated keeping the logic same. Zip align utility can be used to realign the data on .apk file. Zip align utility is available on android SDK and is originally designed for providing optimization for .apk files. Zip align basically realign the uncompressed data within .apk file on 4 byte boundaries so that all the portion can be accessed through map() function. this realignment technique preserve the internal structure of the .apk file but it changes the signature pattern of .apk file such as cryptographic hash of the .apk file. If an anti-virus software directly detect malware based on cryptographic hash such as MD5 (message digest algorithm) then this alignment technique can easily evade the detection of anti-virus system. Malware programmer may implements 8-byte alignment technique instead of 4-byte alignment to actually transform to a different output because, It is usually recommended to use 4-byte boundaries alignment to achieve optimization. Thus 8-byte alignment method ensures that malware is actually transformed to different output.

Resign:Whenever an android developer wants to publish an android app on Google play it is required by android to sign the .apk before publishing and running on smart phone. Citing the official document of android it is allowed to self-sign the android application before publishing. This does not involve any trusted central authority such as VeriSign or go daddy to sign the .apk file. Also .apk file can be signed multiple times. In this way different signature can be generated and attached to .apk file. This actually evades the detection of malware which identifies malware by its original sign.

Tools such as jar signer utilities can be used to resign an application, both utilities are available on android sdk. First key tool can be used to self-sign an android application and store the key in key store, then jar signer utility can be used to re-sign the application .apk file using the stored key results in generating apk with different sign.

Rebuild: The rebuild technique rebuilds the Dalvik byte code using apk tool which results in change byte code order of original Dalvik byte code. Dalvik byte code within an .apk file is first disassembled into small code using apk tool. Then The small code is again assembled using apk tool to byte code the original .apk and repacked .apk are exactly same with the only difference in byte code order. This difference depends upon the parser analysis. Hence the resulting Dalvik byte code (and hence the .apk) is different than original at the same time keeping the logic and functionality of the original .apk same.
After rebuilding process .apk can be re-sign as describe above with randomly generating private keys. This technique can successfully evade anti-virus system that uses cryptographic hash or .apk signature for detection of malware.

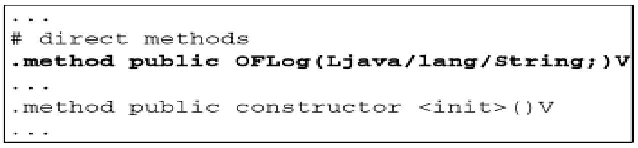
B. Code Obfuscation

In code obfuscation, an obfuscated code is added in .apk file. Program code of .apk is first modified so as to make reverse engineering more difficult. Specifically, it alters the size and contents of .apk file keeping the logical behaviour same. Code obfuscation first is performed on smali code which is assembly-like language based on Dalvik executable code. The debug information about how the variables and method are invoked are provided by smali syntax which makes easy to add an obfuscated code in an .apk file

The obfuscation process involves adding an obfuscated code into .apkwhich is carried out by disassembling .apk filewith the help of apk tool utility. This disassembling results into a smali code onto which an obfuscated code is added. First .smali code is modified by different obfuscated techniques which we will describe shortly.After modification smali code is rebuild into .apk file using apk tools and sign the output of .apk file as described previously using rebuild and resign techniques. Then zipalign tool is used to realign .apk file.

There are various types of code obfuscation method some of which are automated while others require manual intervention. In our case a manual intervention is required because smali code obfuscation relies on underlying semantics of the program. While modifying the code one need to take care of preserving the logic of the code. For example while substituting the code with different code we have ensure that it does not change the behavioural aspect of code. Our aim is to show that malware can be transformed into different variants easily with simple obfuscation techniques that can evade detection of anti-virus system.

Inserting Defunct Method: Signatures that are based on method table of Dalvik byte code can be change by adding defunct method. When these defunct methods are added it does not change the logic of the original source code instead modify the Dalvik byte code table and subsequently change the signature which was based on that table. defunct method can be added in various ways, for example by implementing the Log.d debug method which is simple android utility method to output debugging information. in string format. Log_d method is disassembled into smali format and then it is placed before the constructor method of each class of smali format. The constructor method can be found locating a string '# direct methods' in each smali file.

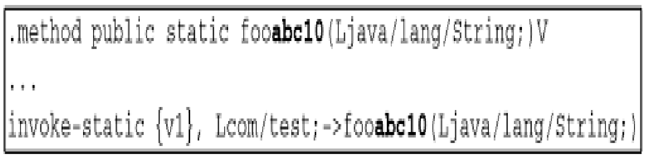


```
...
# direct methods
.method public OFLog(Ljava/lang/String;)V
...
.method public constructor <init>()V
...
```

Fig 3 Inserting defunct method (i.e. OFLog)

Renaming: In this method a randomly generated String (For Example abc10) is appended at the end of user-defined method. In this way the signature which is based on the method name can be changed. To implement this method, first we have to differentiate the system library method names which are found in .jar in android SDK from user-defined method name. After identifying user-defined method we find that method in .smali file and then append the random string with those methods, thus resulting in change in signature which can easily evade anti-virus software.
These renaming techniques can also be applied to different types of identifiers including packages, variables and classes. This make the code more obfuscated and can easily evade from anti-virus detection. Following figure illustrate the renaming technique in which user-define method is renamed into fooabc10.

.

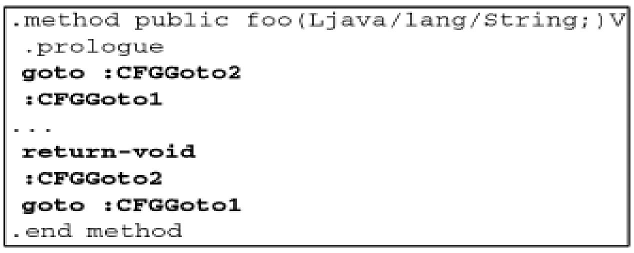


```
.method public static fooabc10(Ljava/lang/String;)V
...
invoke-static {v1}, Lcom/test;->fooabc10(Ljava/lang/String;)
```

Fig.4 Renaming method from foo to fooabc

Changing Control Flow Graphs (CFGs): Control flow graph show the execution path of code on the basis of which signature can be generated. Anti –malware software can use these CFG generated signature to detect malware. CFG signature is based on grammar table. Here we change the CFG signature by modifying the CFG of smali file.

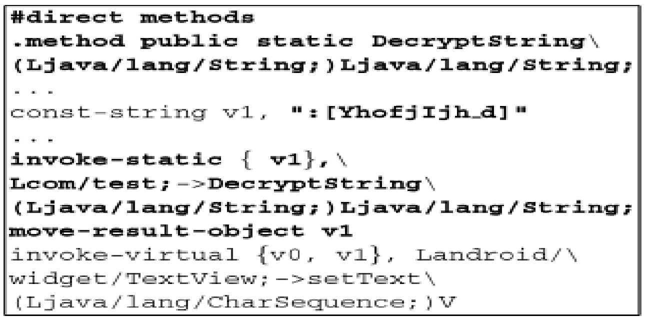
In the following example goto obfuscation is used in which goto statement is inserted at the beginning and end of method. In this way control flows from first goto to second goto. However to avoid the loop condition return statement is added before second goto, so when next time the control flows to second goto, return statement is executed first and the method is finished with returned.



```
.method public foo(Ljava/lang/String;)V
 .prologue
 goto :CFGGoto2
 :CFGGoto1
...
 return-void
 :CFGGoto2
 goto :CFGGoto1
.end method
```

Fig. 5 Go to obfuscation

Encrypting Constant Strings: Constant string in the code is one which cannot be changed and hence it can be used to generate signature. Anti-virus system can use this signature for detection of malware.
But this signature can be easily changed by encryption and decryption of constant string. Here we can implement simple symmetrical method to encrypt a String. We implement Caesar cipher method for encryption in which each character byte is shifted (of each alphabet) by an integer value. For example, suppose if we want to encrypt a string "malware" by integer value of 10 (Key value) the result of encryption process will be "wkvgkbo". Similarly we can decrypt the string by adding all byte by 10 integer value which is the key of cipher. We can implement this process in a Text View control, so before the Text View control is called the decryption method is carried out. When the string is decrypted the signature is changed and it can easily evade the detection of Anti-virus system.



```
#direct methods
.method public static DecryptString\
(Ljava/lang/String;)Ljava/lang/String;
...
const-string v1, ":[YhofjIjh_d]"
...
invoke-static { v1},\
Lcom/test;->DecryptString\
(Ljava/lang/String;)Ljava/lang/String;
move-result-object v1
invoke-virtual {v0, v1}, Landroid/\
widget/TextView;->setText\
(Ljava/lang/CharSequence;)V
```

Fig. 6Encrypting the constant string

VI. Conclusion

We find different types of malwares that existed in mobile devices and how they get undetected by anti-malware software. Malware developers implements transformation techniques such as Repackaging and code obfuscation that can evade detection of malware thus making anti-virus tools ineffective. Also android security model is also subjected to various types of vulnerabilities. Thus Its important for commercial anti-malware software's to be aware of such transformation techniques and also update itself with new techniques that are implemented by malware programmers to evade detection in order to make android platform more secure to use.

REFERENCES

[1] Miller, C.; Mobile Attacks and Defense, Security & Privacy, IEEE, July-
Aug. 2011, pages 68 - 70
[2] Bläsing , T.; Batyuk, L.; Schmidt, A.-D.; Camtepe, S.A.; Albayrak, S.; -
An Android Application Sandbox System for Suspicious Software
Detection. Malicious and Unwanted Software (MALWARE), 5th
International Conference, pages 55 - 62. IEEE 2010
[3] Detection of Intrusions and Malware, and Vulnerability Assessment: 9th ...edited by Ulrich Flegel, EvangelosMarkatos, William Robertson

Mr.SharadR.Kurhade
He is a student and currently pursing MCA (Master of computer application) from IMCOST college thane affiliated to Mumbai University and have done graduation in BCA (Bachelor of computer application)

Mr.Nayan D.Gite
He is a student and currently pursing MCA (Master of computer application) from IMCOST college thane affiliated to Mumbai University and have done graduation in BSc. (Computer Science).

[4] Android Security. [Online]. Available:
http://source.android.com/tech/security/index.html
[5] Forbes Video Network – Intelligent Technology, [Online]. Available:
http://video.forbes.com/fvn/future-tech/vmware-virtual-smartphone
[6] Ubuntu One, [Online]. Available: https://one.ubuntu.com/
[7] Delac, G. ;Silic, M. ; Krolo, J. ; Emerging security threats for mobile
platforms - MIPRO, 2011 Proceedings of the 34th InternationalConvention, pages 1468-1473, July 2011