Android Security Analysis using Mobile Sandbox

Munishka Vijayvergiya, Abhignya Tayi, Sanya Jain, Sanjana Reddy

Abstract: Today, smartphones and Android devices are effectively in development like never before and have become the easiest cybercrime forum. It is necessary for security experts to investigate the vengeful programming composed for these frameworks if we closely observe the danger to security and defence. The main objective of this paper was to describe Mobile Sandbox, which is said to be a platform intended to periodically examine Android applications in new ways. First of all in the essence of the after-effects of static analysis that is used to handle the dynamic investigation, it incorporates static and dynamic examination and attempts to justify the introduction of executed code. On the other hand, to log calls to native APIs, it uses those techniques, and in the end, it combines the end results with machine learning techniques to collect the samples analysed into dangerous ones. We reviewed the platform for more than 69,000 applications from multi-talented Asian international businesses sectors and found that about 21% of them officially use the local calls in their code.

Keywords: The Main Objective of This Paper Was To Describe Mobile Sandbox

I. INTRODUCTION

Smartphone sales have impressively hit the next stage in recent years. The attention of cybercriminals who try to manoeuvre the user into installing harmful software on the computer was drawn to this phenomenal growth. We tried to understand about 6,100 harmful applications in previous work and assembled them with the aid of the VirusTotal API into 51 malware institutions. In addition, 45 percent of our malware institutions sent text messages. Interestingly, these messages were sent to affect individuals such as industrialists, the numbers of politicians to make money immediately.

Until recently, the research was performed manually with the aid of manpower using instruments such as compilers and debuggers with their own intelligence, which takes a lot of time and, depending on the expert's ability, often results in errors. Tools were therefore built.

A static analytical approach, focused on signature detection of applications, serves as a popular antivirus technology approach.

The back or trap door uses the approach in use where the unintelligible methods are made more difficult for static study.

By using the read elf utility, they worked to take out the function calls from an Android application and made the difference in producing the list of familiar malware info. Malware writers created the behaviour of rendering ambiguous methods that have been shown to be successful against changeless research, parallel to the environment of desktop PCs.

References:

II. BACKGROUND

With the notable usage of smartphones and the distribution model of applications, criminals are making their work easy with the help of smartphones as a potential target for malware to steal private information, misuse it for premium SMS services, or try to manoeuvre necessary banking information (mTAN) on these devices. Sometimes, we can even find these threats within one hostile app. In this area of paper, we give a short overview of current mobile threats and describe why and how the Android platform is the most targeted mobile platform. Mobile threats are categorized into two classes: web-based and application-based threats.

These threats depend on the rigorous usage of mobile browsers and their numerous options of usage of implementations.

- malware is the software it is designed to cause damage to the mobile sandbox. It has the potential to wipe out all the data in the mobiles.
- personal spyware gathers personal information and it relays on data firms, advertisers etc. normally it tracks and sells your internal usage data, and captures the credit card and bank account information.
- Grayware operates similarly to malware, but it is not transmitted to harm users directly. It won't affect the functionality of the system. Above all, information about us patterns is collected for the purpose of selling any of this data or to make advertisements in a manner.

References:
2. https://scholarworks.sju.edu/cgi/viewcontent.cgi?referer=&httpsredirect=1&article=1488&context=etd_projects
III. OBJECTIVES

The main aim of our project is to explore the static and efficient ways of using software tools to detect malware on Android, compare and decide the best method. As we all know, cyber crime is rising day by day, so Android is increasingly focusing on it. For researchers to evaluate hateful applications written for programmes, knowing the safety and privacy risks is important. Mobile-Sandbox is a software designed to test Android apps in several ways automatically. It includes static and dynamic analysis, i.e. the static analysis outcomes used to direct dynamic analysis and increase the generated code coverage.

IV. SYSTEM ARCHITECTURE AND IMPLEMENTATION

To search for malicious software, both static and dynamic analysis have been deployed. In order to identify applications as malicious or benevolent, static analysis uses machine learning and these results can be used in dynamic analysis. The programme is performed in an emulator in dynamic analysis and every process is logged in. The system uses some methods to log calls to native APIs, which is an essential process since certain malicious applications can call certain functions outside their native codes, and it is crucial to track all these activities. To give final results, all these outcomes are then combined.

A. Static Analysis:

Static analysis initially checks the app downloaded and its source code. Many antivirus technologies use signature-based detection of apps and a similar approach is used by static analysis. Many malware systems have begun to use certain obfuscation techniques. These techniques make static analysis trickier and can often surpass static analysis without being caught as malicious. One such method used by many android applications is concealing certain dangerous activities. They do this by calling functions outside the Dalvik/Java runtime library. These risky functions may be written in C/C++. This is where Dynamic analysis comes handy.

Firstly, the hash value of the database is checked with a database called VirusTotal. This tells us how many tools have marked this application as malicious or benign. Static analysis is crucial to track all these activities. To give final results, all these outcomes are then combined.

B. Dynamic Analysis:

Dynamic analysis looks at the run-time behavior of an application. This helps to figure out many threats/dangerous activities which may have been missed during static analysis. Dynamic analysis executes the application being inspected in a controlled environment called sandbox. Every relevant operation of the execution of the application such as sending SMS messages, reading data from storage, and connecting to remote servers, is monitored and a report is generated. Though Dynamic analysis handles obfuscation techniques, it is circumvented by dynamic detection methods. This is why, it's important to combine static and dynamic analysis.

For dynamic analysis, the Android emulator provided by Google is used. This emulator can also be set to its previous state once the application has been tested. This is an advantage. The latest version of Droidbox has been used as the basis for Dynamic analyzer to make it compatible with the recent Android versions. Droidbox however lacks the support needed to track native API calls which is why Android NDK has also been used. ltrace command has also been used to track the code. All this information including the native calls required to check shared objects is documented in separate files. Wireshark is a network simulation tool and it has also been used to monitor all the data sent over a network. Most applications require user interaction for them to be assessed properly and Ranorex Studio has been used for the same. It generates a number of interaction elements which are sufficient to track all possible activities of the application.

References:
V. USE CASE DIAGRAM

This figure shows the use case diagram of android security analysis using mobile sandbox

VI. DFD DIAGRAM

Level 0

This figure shows the DFD level 0 diagram of android security analysis using mobile sandbox

Level 1

This figure shows the DFD level 1 diagram of android security analysis using mobile sandbox

VII. EVALUATION

For mobile sandbox system the following aspects are being evaluated:

1. Correctness
2. Performance
3. Detectability

A. Correctness:

Using correctness, the entry to the application i.e Mobile sandbox can only be done if the proper actions are being performed by the analysed app. If these actions satisfy one can access the log file of the mobile sandbox.

Methods to check correctness:

1. We can use data visualisation techniques i.e Machine learning to find the most sophisticated malware using plots such as scatter plot, heat maps and line plot.
2. The sample we chose consists of various families of android malware which is meant to assure the coverage of malicious actions. The sample data is given below.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Detected Malware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adsrs</td>
<td>S</td>
</tr>
<tr>
<td>BaseBrid</td>
<td>I, B</td>
</tr>
<tr>
<td>SerBG</td>
<td>R, I, B</td>
</tr>
<tr>
<td>RootSmart</td>
<td>R, I, B, A</td>
</tr>
<tr>
<td>LeNa</td>
<td>C</td>
</tr>
<tr>
<td>Moghava</td>
<td>S</td>
</tr>
<tr>
<td>Tapsnake</td>
<td>L</td>
</tr>
</tbody>
</table>
The following describes different Android malware families that are intended to ensure malicious activities are protected.

A -> installs additional apps  
B -> Botnet characters  
C -> compromises local storage  
D -> steals private related data  
L -> steals location data  
R -> gains root access  
S -> sends SMS messages

These two methods are used to find the most sophisticated and least sophisticated data.

From the considered data the most sophisticated malware sample is “RootSmart” and the least dangerous Malware family is “TapSmart”.

A. Performance:

This system, i.e., The mobile sandbox is powered by Intel Xeon2. In general, it takes 3s for a mobile sandbox to check viruses in the system and additional 10s - 15s would be required for subsequent static analysis. Cleaning and rebooting a new version takes the machine roughly 3 minutes. It takes a further 4-6 minutes to install and download the application after rebooting. It takes another 8-10min for the programme execution and monkey runner scripts to run. The application's execution time is solely dependent on the application's duration. It takes another 2s to record all the files after shutting down the emulator. This is how success takes place and it is revised from time to time. For performance enhancement or to reduce the time taken, dynamic analysis may be used for improvement.

B. Detectability:

There are mechanisms in the Windows environment to detect sandbox environments and virtualized ones to transform this method into a malicious programme. Detectability plays a major role in the research platform for safety purposes. The following table describes the malware family and the number of samples of malicious data.

We can use machine learning techniques to plot data and understand which malware family is harmful.
Figure 8 describes the detectability scatter plot of the various families of android malware using machine learning techniques.

```
Text(0.5, 1.0, 'detectability plot')
```

Figure 9 describes the detectability linear plot of the various families of android malware using machine learning techniques.

Figure 10 describes the detectability heat map of the various families of android malware using machine learning techniques.

References:

VIII. CONCLUSION

In this research paper, we have proposed an updated version of Mobile Sandbox which uses both static and dynamic analysis to check for threats in mobile applications. Each application is thoroughly monitored and every activity of the application in inspection is checked to expose all possible security threats to the data and the user. Although static analysis focuses primarily on the application’s source code, dynamic analysis runs the application in a structured environment to show all of its operations. In order to ensure no malicious application is left off-guard, even native API calls have been reviewed. As long as the application is on the computer, users can build an account to sign in and from there, it continuously tracks all the apps added during its time span, reviews app permissions and their functions to provide the user and his/her data with a safe and stable mobile environment.

ACKNOWLEDGEMENT

It is our pleasure to express a deep sense of gratitude to our professor Vani V. for her constant guidance and understanding and also having faith in our capabilities to finish the project.

We would also thank our parents for believing in our abilities and also providing an emotional cushion whenever we stumbled. Cheerful due to a number of our classmates without their support this task wouldn't have been completed. Especially we thank God for providing me with new opportunities each time.

REFERENCES
2. Android Developers.: Using the Android emulator
3. Android Developers.: Android platform versions
http://developer.android.com/about/dashboards/index.html
4. Static and malware detection for Android malware detection
https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer=&httpsredirect=1&article=1488&context=etd_projects
https://github.com/floe/ltrace

AUTHORS PROFILE

Abbignya Tayi, is currently a 2nd year student pursuing B.Tech in Computer Science at Vellore Institute of Technology, Chennai. She is self directed and self motivated person. She believes in team work. She is a hardworking individual who takes interest in sports, python development, app development, machine learning, Artificial intelligence, photography and video editing. She takes special interest in cloud computing and have done many labs using quiklab platform. Her research interests include Machine Learning, Network security, Software development, Mobile application and Computer Vision. She aims at working as a Software professional in the IT field at top product based companies.

Munishka Vijayvergiya, is currently a student pursuing B.Tech in Computer Science at Vellore Institute of Technology, Chennai. Her research interests include Android Security, Machine Learning and Blockchain. She is a highly optimistic individual who also takes interest in Public Speaking, Dance and Reading. She firmly believes in the power of hard-work and sincerity at work. She also spends considerable time doing web-development and thoroughly enjoys it. Being an avid reader since her childhood, she also took interest in writing and gained recognition through CBSE expression series during her school time. She aims at working as a Software professional in the IT field while continuing her research-work.