Implementation of Private Base Station in Android Container Environment for Mobile Communication

Jaehyeon Yoon
Department of Electronic Engineering, Soongsil University
Seoul, Korea
yjh7593@soongsil.ac.kr

Jungsoo Park
School of Software Convergence, Soongsil University
Seoul, Korea
ddukki86@ssu.ac.kr

Souhwan Jung
Department of Electronic Engineering, Soongsil University
Seoul, Korea
souhwanj@ssu.ac.kr

***Abstract*—** Recent research discovered a noticeable number of intelligent malicious apps that could use various techniques to avoid analysis and detection. Because of that reason, there is a need for an analysis platform for efficient analysis. Existing works on bare-metal Android analysis such as real devices and Android container are still depending on real-world mobile network, which is costly. In this paper, we propose an idea of using private base station to support mobile features on bare-metal based dynamic analysis system. Our implementations show that private base station can be used to increase productivity in analyzing the intelligent app. With this model, we can solve the disadvantages of the existing analysis platform.

Keywords—private base station, android container, mobile network, intelligent app, dynamic anlaysis

I. Introduction

Today, a variety of mobile devices are being used in our lives, and the Android operating system occupies more than 74% of the mobile market[1]. Various functions such as mobile payment and banking can be performed on the Android device, and some malicious codes are attempting to bill them without informing the user. There are some proposed methods for static and dynamic analysis, but various methods for avoiding the analysis platform can be used. Encryption, dynamically loaded files, and obfuscation functions can be used to make static analysis difficult. Then they determine whether it is a dynamic analysis platform or real device and can execute only normal code. In the case of dynamic analysis using real devices, it takes a lot of time resources to recover the analysis environment after testing one malicious code.

The Android container has dramatically shortened the recovery time by the simple task of shutting down the Android container and copying the template in Host OS. In addition, the QEMU-based Android VM(Virtual Machine), which can be used for dynamic analysis, is difficult to use SMS, and is providing the Internet using virtualized Ethernet devices instead real Wi-Fi. This feature can be the same in the Android container environment, easily exposed by malicious applications. It can make analysis more difficult.

When the code of the application tries to charge via SMS, it can make problems. Several papers have already been warned about SMS billing activity[2,3]. In the VM-based platform, the codes to be analyzed are not performed. And in a real device-based platform, additional monetary spending due to malicious code analysis can occur. To solve this problem, we constructed a private base station using BladeRF and customized the framework to connect to the carrier in the Android container.

Our system can make some contributions as follows.

- We implemented the dynamic analysis system with private base station and Android container for the SMS related functions.
- The proposed system enables the dynamic analysis related to the SMS and the like to be performed without additional charging using a private network for analysis, rather than a mobile network carrier actually used.

II. Related works

In Section II, We describe the existing platforms that can be used for dynamic analysis and the pros and cons of those platforms.

A. Container-based dynamic analysis platform

The Android container has many advantages over other platforms for existing dynamic analysis. According to Chau et al [4]., The container-based Android environment has a much faster deployment speed than the ARM emulator and x86 emulator. Unlike Real Devices, it has the advantage of accessing the Android environment without a debugger connection in Host OS. However, due to the lack of various sensors and communication modules provided by real devices, analysis may be limited.

B. VM-based dynamic analysis platform

VM-based Android is used on a variety of platforms[5]. They can implement a variety of Android virtual devices as VMs on x86 based servers. However, the VM-based Android environment has a variety of features to guess the VM environment, such as having an Ethernet interface or */dev/qemu-pipe* and can be easily bypassed by checking at the application level[6]. In addition, instruction transformations are required to run .so library files compiled for the ARM architecture on the x86 architecture, and intelligent malware can catch these features.

C. *Real Device-based dynamic analysis platform*

Dynamic analysis platform based on real device can analyze the app, but has disadvantages in environmental recovery. After the malicious code analysis is finished, the environment must be initialized for later analysis. Generally, in order to initialize one device, it is necessary to access Android's recovery mode and overwrite the changed image with the initial image. In BareDroid[7], application analysis and recovery work are separated and processed in parallel in a real terminal, shortening the time consumed for environment recovery in a real terminal. However, a debugger is required to dynamically access, and an application can easily detect the dynamic analysis environment through debugger detection.

As mentioned above, recent malicious codes are becoming intelligent and using various environment detection and analysis avoiding techniques to prevent them from being analyzed. In order to analyze these malicious codes, a dynamic analysis platform that has an environment similar to a real device and capable of efficient analysis is needed.

III. Proposed Architecture

Android containers have many advantages over other platforms in dynamic analysis. However, the Android container developed on the reference board lacks a variety of function which is in the actual device. (mobile communication, sensor module, etc.) In this situation, if an app requests a specific service (SMS service, telephony service, etc.) to the Android framework, the service may fail to find the proper device and cause the crash. The crashes of service should be resolved because it makes it impossible to analyze the specific codes of app. We built a private mobile network to connect the Android container through USB dongle to make the Android container an environment similar to the real device. We also added a Wi-Fi module to the Android container to connect to the Internet. *"Fig. 1"* shows an overview of the analysis environment using the Android container and the private base station.

The Android container recognizes the USIM device after booting and starts the Telephony Service. The initiated Telephony Service retrieves the surrounding mobile network carrier and connects it to the base station that we implemented.

If a specific SMS is sent by the application, the base station receives the SMS and sends an acknowledgment. The application can successfully execute the code after the SMS transmission, thereby extending the scope of the application dynamic analysis. We also used Wi-Fi to connect the container to the Internet. Since the Internet connection over the mobile network is written to the Android framework, it is likely to be detected by the user. For this reason, malicious applications may run after checking the Wi-Fi connection status for code that causes large traffic. In addition, the existing emulator made it look like a Wi-Fi connection using an Ethernet interface, but we designed the actual Wi-Fi module to use the same signal strength information as the actual device.

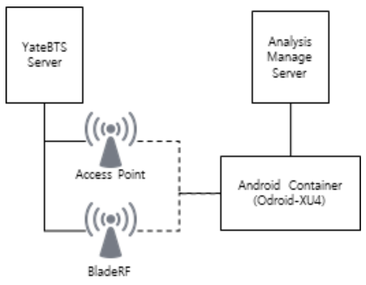


Fig. 1. Overview of proposed system

IV. Implementation

A. *Mini base station*

BladeRF can be used as a device for implementing private base stations. We implemented a mobile network using 'BladeRF x40' and yateBTS[8] to build the system shown in *"Fig. 1"*. yateBTS was implemented on the server and connected to BladeRF via USB. We used the ubuntu 16.04 as OS on server where yateBTS is installed. *"Table I"* shows the options configured with YateBTS and some of the set values.

B. *Android Container*

We used the Android container implemented in Odroid-XU4 and used the HUAWEI e160 USB dongle for USIM connection and USB Wi-Fi module for Wi-Fi connection. The Android images that can be used in the Odroid-XU4 have been made using the community-shared source code. However, the source code is set to be compiled into the tablet version and is customized for Odroid-XU4 unlike AOSP. This causes problems for the Android framework to get the actual IMSI and IMEI values. We modified RILD and Telephony manager in android framework to connect the mobile network. In order to use the HUAWEI USB dongle in the Android container, we have set up an environment for device recognition in the host OS, and compiled, applied, and configured the kernel module to utilize the USB device when booting Android.

We also installed additional APs for the Wi-Fi connection of the Android container and set Wi-Fi information to connect to the */data/misc/wifi/wpa-supplicant.conf* file. The wlan0 device has been compiled with the i802 related kernel module, along with some settings to be passed from the host OS to the Android container and applied to the Android container. Since the Android container does not automatically run Wi-Fi at boot time, we changed the Wi-Fi state via the *svc wifi enable* command to the Android shell after booting.

TABLE I. CONFIGURATION OF YATE BTS

Options	Values
Radio.Band	900
Radio.C0	75
Identity.MCC	001
Identity.MNC	01

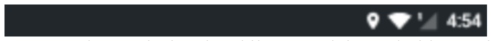
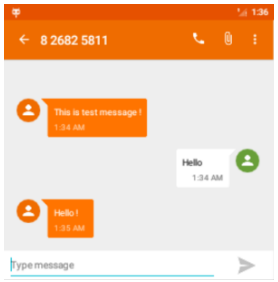


Fig. 2. Icon about Wi-Fi and mobile network in Android Container



8 2682 5811

This is test message !
1:34 AM

Hello
1:34 AM

Hello !
1:35 AM

Type message

Fig. 3. SMS communication of Android Container

C. *Implementation results*

The Android container has been templated and stored since its initial configuration. It can make the android container to easily boot and shut down. After booting the Android container, you can see that the Wi-Fi and mobile networks are connected normally as shown in *"Fig. 2"*, depending on the configuration.

Android container transmits device information such as IMEI, and the private base station allocates and connects a telephone number using the information. The short message service center in yateBTS enables the SMS sent by the Android app to communicate with other devices. "Fig.3" shows the result of sending and receiving SMS in the Android container.

V. Evaluation

The evaluation of the proposed system was performed as follows. The dataset for evaluation was constructed as follows. 50 AMD malware apps, 25 financial apps, and 25 environment detection apps. We used three platforms to see how much additional dynamic analysis was performed. The dynamic analysis evaluation performed on each platform is shown in "*Table II"*, and each number indicates how many apps were executed on the platform without crashes.

"TABLE III" compares the features of the proposed system with the existing analysis platforms. The rooting detection feature is whether the app can run if it detects the root environment. Because the x86 emulator can control the root environment with settings, it can vary depending on the configuration. The emulator detection feature indicates whether the app is executable when it detects the emulator environment. The SMS related function expresses whether or not the app operates normally without crashing when it wants to execute APIs related to SMS.

VI. Conclusion

We implemented a mobile network and applied it to the Android container to configure the Android container as a real environment for app analysis. The implemented network system works well for the Android container, and the app running in the Android container showed that there is no problem in SMS and internet connection using this network. This study presents and implements an environment that can perform more effective analysis compared with the existing Android app dynamic analysis platform.

TABLE II. NUMBER OF ANALYZED APPS WITHOUT CRASH

Environment	**# of analyzed apps**
Rooted phone	53/100
Emulator	50/100
Proposed system	93/100

TABLE III. COMPARISON OF THE FEATURES WITH THE EXISTING PLATFORM

Features	**Real device**	**Rooted device**	**x86 emulator**	**Android container**	**Proposed system**
Rooting detection	X	O	Depend on config	X	X
Emulator detection	X	X	O	X	X
SMS related features	O	O	X	X	O

VII. Future work

As a future research, there may be a system implementation for analyzing an app through SMS trigger using the implemented private base station. It will be able to dynamically analyze the code that is triggered from SMS and extend code coverage in dynamic analysis.

Acknowledgment

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP), a grant funded by Korea government (Ministry of Science and ICT) (no. 2016-0-00078, Cloud Based Security Intelligence Technology Development for the Customized Security Service Provisioning) and Institute of Information & Communications Technology Planning & Evaluation (IITP), a grant funded by Korea government (Ministry of Science and ICT) (no. 2017-0-01853, Machine Learning based Intelligent Malware Analysis Platform).

References

[1] "StatCounter Global Stats." Internet: http://gs.statcounter.com/os-market-share/mobile/worldwide, Mar, 2019[Mar, 15, 2019].

[2] L. MIN and Q. CAO, "Runtime-based behavior dynamic analysis system for android malware detection," *In Proceedings of the 2012 2nd International Conference on Computer and Information Application*, 2012, pp. 233-236,

[3] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," *2012 IEEE symposium on security and privacy. IEEE*, 2012. p. 95-109.

[4] N. T. Chau, and S. Jung, "Dynamic analysis with Android container: Challenges and opportunities," *Digital Investigation*, no.27, pp.38-46, Dec, 2018

[5] Joe LLC, " Automated Malware Analysis - Joe Sandbox Mobile," Internet:https://www.joesecurity.org/joe-sandbox-mobile, [Mar, 1, 2019]

[6] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection," *In Proceedings of the 9th ACM symposium on Information, computer and communications security. ACM*, 2014, pp. 447-458.

[7] S. Mutti, Y. Fratantonio, A. Bianchi, L. Invernizzi, J. Corbetta, D. Kirat and G. Vigna, "Baredroid: Large-scale analysis of android apps on real devices," *In Proceedings of the 31st Annual Computer Security Applications Conference*, 2015, pp. 71-80

[8] "Android Malware Dataset," Internet: http://amd.arguslab.org/, [Feb, 2, 2019].