Received October 13, 2020, accepted October 30, 2020, date of publication November 4, 2020, date of current version November 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035774

A-Pot: A Comprehensive Android Analysis Platform Based on Container Technology

JUNGSOO PARK[1], NGOC-TU CHAU[2], LONG NGUYEN-VU[2], JAEHYEON YOON[2], AND SOUHWAN JUNG[3]

[1]Department of Software Convergence, Soongsil University, Seoul 06978, South Korea
[2]Department of Information and Telecommunication Engineering, Soongsil University, Seoul 06978, South Korea
[3]Department of Electronic Engineering, Soongsil University, Seoul 06978, South Korea

Corresponding author: Souhwan Jung (souhwanj@ssu.ac.kr)

This work was supported by Institute of Information and Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00477, Development of Android security framework technology using virtualized trusted execution environment).

ABSTRACT Recently, intelligent Android malware avoids being analyzed using anti-emulator, anti-debugging, and rooting detection. Existing emulators have problems to be easily detected by malware that check with hardware or sensor information. This paper proposes an efficient analysis system A-Pot, to deal with intelligent Android malware. A-Pot applied the Android container technology. It's made to be similar to a real phone using ARM-based hardware. A-Pot is equipped with sensor modules such as USIM, Bluetooth, and Wi-Fi module. In order to respond to the environment analysis, the properties of the Android OS are made to be the same as the real mobile phone. In addition, A-Pot is designed to connect to a mini base station for supporting SMS and phone calls with the 3G network. Moreover, with the advantages provided by the container technology, A-Pot is able to support non-ADB, non-debuggers, and non-root environments. To prove the efficiency of our platform, we analyzed using intelligent Android malware, antivirus, Google Play apps, and general malware. This model had an operating rate of about 97.36% for 5000 malware. The proposed A-Pot can be efficiently applied to defend against intelligent Android malware analysis.

INDEX TERMS Android security, malware analysis, intelligent malware, android, evasion techniques.

I. INTRODUCTION

Google analyzes the apps registered in the Play store using the Google bouncer [1]. Malware detection rates increased about 40% before and after the introduction of Google Bouncer. Google also provided another solution, Google Play Protect, to support the verification of released applications [2]. Google Play Protect using AI technology. It can analyze 50 billion apps a day. However, even with Google Play Protect, malware apps still cause damages due to post-detection. In December 2018, 22 apps with more than 1 million downloads on Google Play Store were classified as malicious and deleted [3]. There are various technologies related to malware analysis, but there are intelligent malware that cannot be analyzed due to the limitations of the analysis environment. For this reason, the development of a tool for dynamic analysis of Android is an important research subject. For Android static analysis, we provide AndroidManifest.xml file analysis and source code analysis [4]–[6]. Android dynamic analysis is conducted using either an real device or an emulator [7]–[9]. When using real devices, it is possible to cope with malware that checks hardware, sensor, and CPU information. However, it is difficult to construct an environment for analysis, and initialization takes a long time following the analysis of an application. In the case of the emulator analysis, environment configuration, such as construction and initialization for analysis, is efficient; however, it is difficult to cope with malware that identifies various environments unlike real devices. Recent intelligent malware not only verifies environmental information, but they also send SMS to verify their functionality, and various methods are being used to evade the analysis environment such as debugger detection and root detection [10]–[12]. And a lot of research on how to deal with intelligent malware have been done, but each scheme has some limits. Therefore, we propose A-Pot to detect intelligent Android malware. A-Pot utilizes an Android container-based emulator to build an environment similar to a real phone, and thereafter perform dynamic analysis. The contributions of this study can be summarized as follows: (1) We built an analysis environment similar to real phones by implementing Android container technology on ARM hardware. (2) Sensor modules

The associate editor coordinating the review of this manuscript and approving it for publication was Ilsun You.

such as Wi-Fi and Bluetooth are connected to the Android container. And the build process is configured to emulate a real device so that the return value for various environment search results can be the same as that of the actual phone. In addition, it is built to enable SMS and phone calls using the mini base station. In non-root environments, we utilize the hooking technology without the debugger, and we can analyze not only the JAVA area but also the system call in the native development kit (NDK) area. (3) It operates in the rooting environment, emulator environment, and A-Pot environment for many apps, and confirms the operation rate. It is also designed to cope with activity-aliases. The rest of this paper is organized as follows: We briefly introduce the background of the Android malware analysis technique in Section 2 and describe the architecture of the system in 3. In Section 4, we describe the implementation environment and contents. In Section 5, we discuss and evaluate the performance. Finally, we conclude our findings and state the limitations of our study in Sections 6 and 7.

II. RELATED WORK

A. PROBLEMS WITH ANDROID MALWARE DYNAMIC ANALYSIS

For the dynamic analysis of Android malware, various virtual environments (emulators) are designed, or analysis systems using a real device is deployed. The analysis environment can be largely divided into Bare QEMU, a real device incorporating an analysis tool, and Intel CPU-based Android virtual machine (VM). Examples of Bare QEMU-based malware analysis technologies are Android virtual device (AVD), Blue Stack, AMIDuOS, and Nox. QEMU is an emulator and virtualization tool that can be used to virtualize the hardware of an AVD. With QEMU, the emulator can configure a virtualized environment [13]. However, with QEMU, it can be difficult to analyze the malware, because the malware can easily recognize the emulator environment by checking directories such as /dev/qemu pipe, /dev/socket/qemud or QEMU inspection command through the getprop command. Intel-based Android VMs can be analyzed in notebook and PC environments, and there are Android-x86 types such as Remix OS [14]. Because Intel-based Android is provided as ISO (Operating System Image) files, it can run on virtualization tools such as Virtualbox or VMware. But it is easily evaded by malware that detects virtual environments. There are various strategies to circumvent dynamic analysis; the following strategies are used to determine the virtual environment [15].

It is noteworthy that using Google Bouncer, in 2015, more than 1 million users were harmed by malware using *build.MODEL == "google_ sdk"*, the source code of the Brain Test malicious apps that bypassed Google Bouncer [16]. In addition, in the case of the Intel CPU-based emulator, analysis is difficult due to problems such as frequent interruption of execution, avoidance of analysis when an application is executed, slower speed than a real device, and the impossibility of executing an SO file. Because of these issues, analysts conduct analysis using real devices, including analysis tools. However, the actual device must be a rooting environment. In addition, it needs to connect to ADB for analysis. It is difficult to analyze malware using analysis evasion techniques.

TABLE 1. Emulator evasion technology.

Evasion technology	Description
System	Detection of terminal ID, phone number, build value, QEMU, and similar components.
Sensor	Check the acceleration sensor or gyroscope return value
Data	Check factors such as the phone number, call, SMS, and battery consumption.
Network	Check the network properties, such as Ethernet and routing.
User input	Receive user input for activation.

B. RELATED PAPERS AND RESEARCH

There are various on-going research endeavors on the analysis of Android malware. This section introduces existing research on Android malware analysis and contemplates its applicability to intelligent Android malware [17]. Previous studies have used on-device, off-device, and distributed analysis methods for malware analysis.

1) ON-DEVICE

This is a method of analyzing an Android phone using a real terminal. However, the challenge of the on-device analysis is that it is difficult to analyze the file system because it is difficult to hook up the system to monitor the file systems or perform network access without root authorization.

2) OFF-DEVICE

This involves simple signature-based detection, which corresponds to static analysis. Although signature-based detection is similar to the current practice of many companies, static analysis hampers source code analysis by obfuscation and packing. The efficiency of static analysis is hindered by its various interrupting techniques such as junk code generation or the insertion of code or goto command; sequence change between opcode functions; package, class, and method name change; and string encryption. Thus, many studies have adopted a method of detecting malware by performing dynamic analysis.

3) DISTRIBUTED ANALYSIS METHOD

This analysis method is conducted partially on the real terminal and partially on the emulator. The analysis results are then integrated. Although this analysis method can be effective, it is vulnerable to a timing attack that confirms the time of the operation. The heap and stack status information of the Android running on the emulator is sent to the real terminal, and traffic and time delay occurs in the process of returning the result by operating the real area such as NDK in the real terminal. In this case, it will not function.

Therefore, various studies have chosen instead to attempt to construct an emulator environment and analyze Android malware to avoid this situation.

TABLE 2. Feature availability of intelligent malware (O means it is support to feature, X means it is not suuport to feature.).

Category	Rooting	Build Proc	GPS	SMS	Call	Bluetooth	Wi-Fi
DroidBox (Emulator)	O	X	X	O	O	X	X
Droid Scope (Emulator)	O	X	X	X	X	X	X
TaintDroid (Real Phone)	X	O	O	O	O	O	O
Andrubis (Emulator + Real Phone)	X	O	O	O	O	O	O
SanDroid (Emulator)	O	X	X	X	X	X	X
Paranoid Android (Emulator)	O	X	X	X	X	X	X
Cooperdroid (Emulator)	O	X	X	X	X	X	X

4) DroidBox
The DroidBox runs the APK file in a sandbox environment, monitors its activity, and then displays the result to the user [18], which may include incoming / outgoing network data, file read and write operations, confirmation of file information outgoing via network or SMS, checks for dangerous permissions, and SMS and call log checks. However, a DroidBox emulator can easily be verified in the process of examining the Android build process.

5) DroidScope
DroidScope is a dynamic analysis Android framework based on VM Introspection (VMI) [19]. DroidScope monitors the OS and Dalvik status using the emulator. It is characterized by the detection of elevated privileges for the Android kernel. Based on the QEMU emulator, the Android malware variants, DroidKungFu and DroidDream, were analyzed and detected using DroidScope. However, the effect of DroidScope on other malware products has not been proven. Furthermore, it has the disadvantage of being easily analyzed for QEMU emulator detection.

6) TaintDroid
TaintDroid monitors applications that leak sensitive information inside the phone in real time [20]. It performs the dynamic taint analysis to track the flow of data in real-time. However, it tracks only some sensitive data, such as phone status and location information, because it has a large overhead handling the dataflow in its entirety; it can also affect the battery consumption and speed of the smartphone. Another limitation of the TaintDroid is that although it can trace the flow of sensitive data and detect an external leak, it cannot block this flow. Because it operates in a real terminal, various dynamic analysis environments can be searched. It also has the disadvantage of requiring rooting.

7) ANDRUBIS
Andrubis is a web-based malware analysis platform that provides analysis results using DroidBox, TaintDroid, APKtool, and Androguard [21]. Users can submit suspicious apps via a web-based interface. Andrubis analyzes the applications on the remote server and returns detailed static and dynamic analysis reports to the web page. However, the platforms it is comprised of, such as DroidBox and TaintDroid, already have their individual limitations; thus, Andrubis cannot be the perfect solution.

8) SANDROID
Sandroid performs the static analysis of authorization, component, malware class, camera, and call [22]. Through dynamic analysis, it monitors file operation, network behavior (for instance, GET / POST request, TCP / UDP connection), and personal information leakage while an application is running, including file operations. Sandroid performs emulator based analysis and has been proven to be bypassed by CPU checks.

9) PARANOID ANDROID
Paranoid Android is a method of detecting abnormal behavior that occurs in the terminal by transmitting and analyzing the responses that occur during every execution of the terminal to the external server [23]. A proxy is used to quickly transfer network traffic between the terminal and the external server. The terminal exports the data, and the external server analyzes the received data to identify the security vulnerability. The external server executes an emulator of the terminal and detects abnormal behavior by applying various abnormal behavior detection methods utilizing components such as memory use and system call.

10) COOPERDROID
Cooperdroid is a VMI -based automated dynamic analysis system designed to aid a better understanding of the behavior of malicious apps [24]. It is a system that performs system call center dynamic analysis for Android apps using VMI; it proves that system call center analysis can effectively detect malicious behavior. However, a method for identifying the virtual environment of CooperDroid has been disclosed by Vidas *et al.*

III. A-POT ARCHITECTURE
We propose the A-Pot system to solve the difficulty of Android malware analysis described in Section 2. The A-Pot system was developed to make it possible to analyze applications using evasion technology through a dynamic analyzer. It is similar to the real device and easily determines the analyzed result. Figure 1 shows the architecture of A-Pot. The server includes a web server, DB, static analyzer, Software AP. ODROID performs dynamic analysis and sends the results to the server. The mini base station contains a

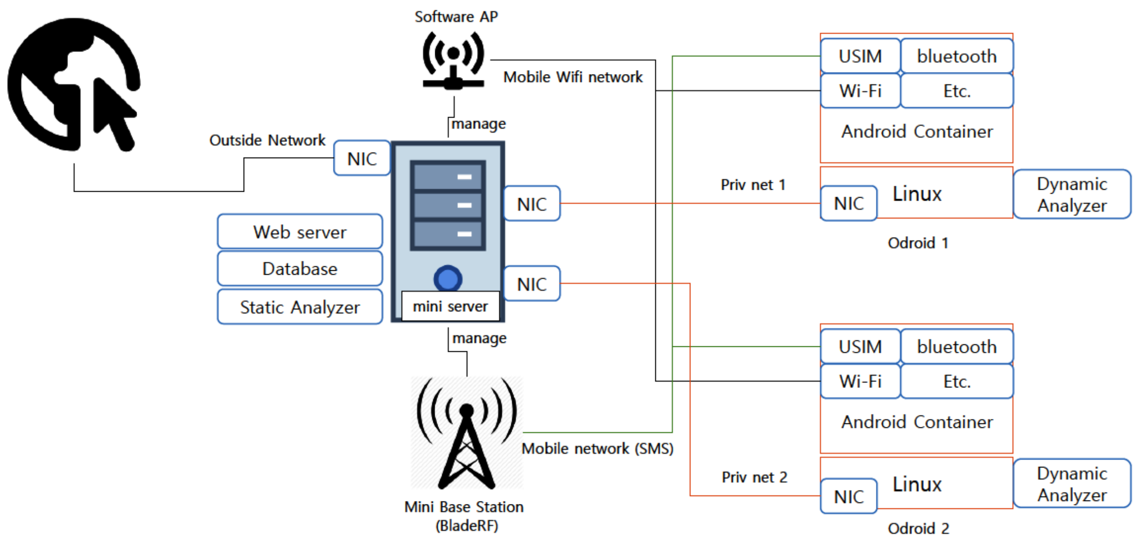
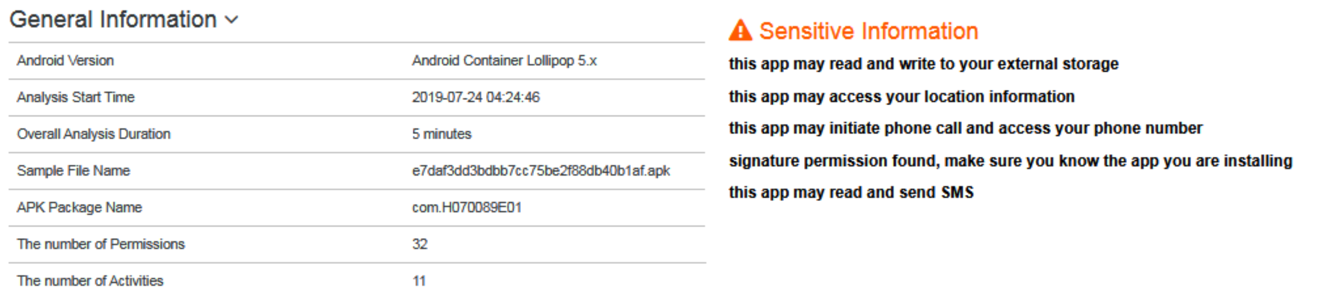


FIGURE 1. A-Pot architecture.

module to handle 3G network and SMS. Network processing is performed through the USIM connected to ODROID.

Some intelligent malware evades analysis through string searches like Frida and Xposed. Therefore, A-Pot was converted and applied to the string of Xposed. When A-Pot analyzed Vulnerability analysts, the characteristics of Xposed were not found. Based on existing research, we've done API hooking that was often used for Android malware. Furthermore, the hooking of the system call of the NDK area is conducted by utilizing the advantage of the ARM-based hardware. It hooks 164 in the Java layer and 35 in the NDK layer to provide the result. In addition, dynamically loaded files are stored by checking for changes in the inode value of a specific folder. If a dynamically loaded file has the DEX, SO, APK, and Zip extensions, it provides additional analysis results on the file. ODROID is connected to the Wi-Fi, USIM, Bluetooth, GPS, and camera modules in dongle form. Sensors respond to intelligent malware that performs a variety of sensor tests. When the sensor is connected, it attaches itself to the Linux environment. In the Linux environment, the attached sensor module is detached, and then attached to the container, so that the sensor can be detected on the actual Android OS. Some intelligent malware may detect Ethernet networks, Ethernet is only connected to the Linux OS, and is used to communicate with external networks; it does not use Ethernet in the Android container. We used Huawei e160 USB dongle and USB Wi-Fi module as a network. We installed additional Wi-Fi for the Wi-Fi connection of the Android container, and programmed the Wi-Fi information to connect to *misc/Wi-Fi/wpa-supplicant.conf* file. The wlan0 device was 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. Because the Android container does not automatically launch Wi-Fi during booting, we changed the Wi-Fi state via the *svc Wi-Fi-enable* command after booting. With this function, IP management for the Wi-Fi connection from the ODROID to the access point (AP) is performed. To monitor network communication through Wi-Fi, the network traffic generated in the Android container can be monitored by installing the Software AP in the Mini server. We use GPS, Bluetooth, and Camera for ODROID provided by Hardkernel without any connection. The mini base station is installed in the mobile base station using the Yate BTS program and BladeRF. BladeRF is widely used as a device for implementing mini base stations. We implemented a mobile network using nuand BladeRF x40 and yateBTS for system construction. It is operated by 3G and connected with the 3G module of Android container for SMS processing and other functions. Two ODROIDs are connected because one user performs automatic analysis continuously, and it is possible for the user to manually input data, receive input, and perform an action in one ODROID. We attempted to circumvent the code coverage, the limitation of dynamic analysis, and to provide various conveniences to the analyst emulating the screen of the real phone. The mini server includes the Apache2 web server and MySQL for user management, and MongoDB for managing analysis results; it performs static analysis, NFS server and dynamic daemon operations. For the dynamic daemon, the command is achieved through the execution of the daemon for dynamic analysis. Static analysis is performed while uploading the apk, but the dynamic analysis is difficult because it needs to additionally check the status of the ODROID. Therefore, it operates as a separate daemon process. A thread is created every 20 seconds, and the MySQL query is used to see if there is an APK file that needs to be analyzed. Then, after checking whether there is a resting host if resting host and APK are found, it is necessary to change the state of the host to analyze mode, create the Android container, and execute the command. ODROID uses the container template to create a container with the name of the APK hash value, and invoke the dynamic analysis trigger API on the web. The



General Information

Android Version	Android Container Lollipop 5.x
Analysis Start Time	2019-07-24 04:24:46
Overall Analysis Duration	5 minutes
Sample File Name	e7daf3dd3bdbb7cc75be2f88db40b1af.apk
APK Package Name	com.H070089E01
The number of Permissions	32
The number of Activities	11

Sensitive Information

this app may read and write to your external storage

this app may access your location information

this app may initiate phone call and access your phone number

signature permission found, make sure you know the app you are installing

this app may read and send SMS

FIGURE 2. General information.

APK between the ODROID and A-Pot server and the internal network file system (NFS) server are used to transmit the dynamic analysis result information.

IV. IMPLEMENTATION

Our analysis server and mobile network server used INTEL NUC Kits NUC8i3BEH and have 1TB of HDD capacity is 8GB RAM. ODROID, a dynamic analyzer, is ARM-based hardware for Android and Linux, provided by Hardkernel; it uses ODROID XU4 version [26]. ODROIDs use the *Samsung Exynos5422 Cortex -A15 2Ghz and Cortex -A7 Octa core CPUs and eMMC5.0 HS400* flash storage. For the mini base stations, bladeRF x40, which is capable of full-duplex 40 MSPS 12-bit quadrature sampling, is used. It can also be easily applied to Linux, Windows, Mac, and GNURadio software [27].

We used 1000 Android malware datasets (AMD) and 1136 Google Play Store apps to measure performance and malware running rates [28]. We used VirusTotal to scan the downloaded app in the Google Play Store. VirusTotal provides an API for scanning services. If all the virus scanners of VirusTotal adjudged the app to be good, it was regarded as a normal app and used. We also received and analyzed 5000 malicious apps from NSHC, a Korean malware analysis company. A-Pot provides results using a separate dashboard to provide analysts with efficient analysis results. First, it displays the APK name, the user name, the size, and the MD5 hash value that was uploaded. Furthermore, it describes the start time and end time, and the operation status is visually displayed on the right side of the dashboard. Figure 2 shows the general information of the analysis result. General information displays the Android version, start time, overall analysis duration, uploaded APK name, sample file name, APK package name, and shows the number of permissions and activities. In addition, it reports the sensitive information and helps the user to intuitively judge it. It also provides risk scores based on risky APIs, permissions, and dynamic analysis. The basic analysis shows the results parsed from the static analysis. Our static analysis parses and lists the main activity, package name, minimum version, target version, version name, receiver list, service, and permissions. In addition, the contents of the code feature are provided in the static analysis process.

TABLE 3. Intelligent malicious code operation rate.

Evasion Scheme \ Tool	Rooted Phone	Emulator(NOX)	A-Pot
SU Detection	0/15	0/15	15/15
Emulator Detection	48/48	13/48	47/48
Debugger Detection	14/65	32/65	63/65
SU Require	6/7	5/7	0/7
Total Operation	53/100	50/100	93/100

TABLE 4. Intelligent commercial app operation rate.

Commercial App \ Tool	Rooted Phone	Emulator	A-Pot
Banking	9/50	8/50	48/50
Vaccine	25/25	24/25	25/25
Game	24/25	18/25	25/25
Total Operation	58/100	50/100	98/100

Code feature analyzes and provides codes with predefined characteristics from the source code of the APK file. It analyzes the Dex binary file at the source code level and makes the characteristics of the code parsable. Next, the APK file is unpacked using Jadx. Then, a search is conducted for the predefined pattern in the extracted java file. Patterns are defined as regular expressions, and the matching patterns are expressed in JSON format and stored in MongoDB. Flow analysis analyzes the Dex binary file at the opcode level and provides a flow graph of the code running in APK. It can show the caller-callee relationship, and display information about class and method values. The dynamic analysis shows the results of the APK uploaded at runtime using A-Pot. To automate dynamic analysis, Monkey is used to generate input to interact with the installed APK, and user interface (UI) Automation is a way to efficiently provide a selection for automated analysis through GUI analysis results. In addition, in the case of manual input, A-Pot is implemented such that it can be used when the user directly inputs the input value and checks it step by step. This process is performed by the user. In the case of Java file hooking, we utilize the technique of installing and hooking Xposed in an unrooted state. The values needed for commencing dynamic analysis are required for the IP of the android container: the dynamic analysis alarm time setting, the APK file path, the package name obtained from the static analysis, and the MainActivity value. After that, the APK Engine concludes the analysis process by storing the analyzed data in the Report Database. Finally,

TABLE 5. Google play store app operation rate.

Operation \ Tool	Normal Phone	Rooted Phone	Emulator	A-Pot
Operation	1118/1136	886/1136	778/1136	1105/1136
Rate	98.41%	85.52%	75.10%	97.27%

the screenshot captures the screen in the middle of dynamic analysis to show if the actual operation was successful. In the case of malicious APKs such as Ransomware, even after the screen is locked, screenshots can be distinguished even if they do not perform very well.

V. PERFORMANCES

In this section, we examine the possibility of analyzing intelligent malware and commercial apps containing evasion technology using A-Pot. We have tested the performance of A-Pot using 100 intelligent malware. These malware were collected after investigating all the ructions of more than 5,000 malware and selected intelligent ones that uses evasion technologies. Our study shows that less than 2% of malware use the intelligent evasion technologies. We categorized the 100 malware according to their evasion schemes as in Table 3 and tested whether those malware can be analyzed in 3 different environment such as rooted phone, emulator(NOX), and A-Pot. Table 3 shows our experimental results that A-Pot outperformed the other two. 93% of malware were analyzed in A-Pot in comparison to 53% in rooted phone, and 50% in emulator. Table 4 shows commercial apps with analysis evasion technology. Various security functions such as are found on banking apps, vaccine apps, and game apps are applied, and various analysis evasion techniques are applied to prevent analysis by hackers. In an intelligent commercial app, 98% of apps ran when A-Pot was used. In the case of banking apps, there was an application that required fingerprint recognition; however, it was difficult to respond to because the fingerprint sensor was not applied to A-Pot.

To get the statistics of normal apps running in different environments, we crawled 1136 app samples randomly from Google Play Store and tested whether they are running normally. Table 5 shows that A-Pot performs like real-phone compared to the others.

Table 6 analyzed the analysis rate of 5000 malicious apps received from the malicious code analysis company in order to judge the operation rate of the application through the malicious code data sample including the intelligent malware. Of the 5000 apps, 4868 functioned normally, and the remaining 132 apps did not work, and could not be analyzed. The reasons for the failure varied; however, only 23 apps were not analyzed because of the version problem; the remaining 109 apps were unusable apps that could not be installed on a real phone. There were various reasons why some apps could not be installed. The first is that some applications cannot be installed due to invalid APK. The second is that it might not be possible to install is the nonexistence of the certificate and signing key of the app does not exist. The third reason is that the AndroidManifest.xml file format is incorrect due to a manifest error. The fourth, if custom permissions overlap, installation is impossible. Fifth, some apps cannot be installed due to a version problem or validation failure during DEX optimization because of a DEX OPT error. Sixth, some apps cannot be installed because the APK is unparsable; thus, it cannot be parsed during app installation. Finally, there are various cases such as the crash of an app when executing activity and service, and the absence of the command and control (C&C server). Consequently, 99.5% of the analyzed apps can be operated, and the majority of the malware operate normally in A-Pot.

TABLE 6. Malware data sample operation rate.

Category	Operation	Fail	Invalid APK
Count	4868/5000	23/5000	109/5000
Rate	97.36%	0.46%	2.18%

To respond to malicious behavior using dynamic loading, a characteristic of malware in recent years, it is determined whether dynamic loading is used or not, as shown in Table 7. Dynamic loading refers to the technique of loading another APK file or Sub-DEX file during the execution of the app, or to the performance of malicious behavior in the NDK area by loading SO file during the execution of the app. Indeed, various malware perform malicious actions through NDK calls, and ARM-based hardware is an essential element for analyzing them. It has been confirmed that normal analysis can be performed, even when dynamic loading is performed in the A-Pot environment. When there is no invocation by a specific trigger among the dynamically loaded unused apps, it is determined whether dynamic loading-related API is used through the static analysis, and the exact result is verified. In addition, packer and obfuscator was applied to some of the 5,000 apps to confirm the result of the dynamic analysis, even though static analysis was not performed properly. Because A-Pot specializes in dynamic analysis, it can obtain results for all apps with obfuscation and packing. Of the 5000 apps, packer and obfuscator were applied to 229 apps. The categories and detailed analysis results are shown in the table below.

VI. LIMITATIONS

Although A-Pot, our proposed platform, is effective, in theory, there are some limitations that an attacker can exploit.

1) The first is the logic bomb. As with previous studies, A-Pot also has no countermeasures against logic bombs. For dynamic analysis, even using manual input for about 5 minutes, it is difficult to determine whether the proper logic bomb worked. Google use Play Protect to counter these logic bombs. However, A-Pot uses

TABLE 7. Rate of apps using dynamic loading.

Operation App	Category	Detail	Count	Total	Rate
4868	Using the dynamic loading	APK, DEX dynamic loading & NDK call	478	2573	52.9%
		APK, DEX dynamic loading	890		
		NDK Call	1250		
	Not using the dynamic loading	Using SDK main DEX	2295	2295	47.1%

TABLE 8. APK with Packer and obfuscator.

Category	Tool	Count	Total
Packer	APKProtect	92	186
	Bangcle	15	
	Ijiami	13	
	Jiagu	18	
	Tencent	32	
	Naga	14	
	Qihoo 360	2	
Obfuscator	DexProtect	31	43
	DexGuard	12	

a separate analyzer, crowdsourcing cannot be used; therefore, it is difficult to circumvent the logic bomb.

2) Second, we did not review malware detection rates. The purpose of this paper is to study the dynamic analysis rate of malware. Therefore, the aspect of android malware detection rate was not mentioned separately.

In existing research on advanced malware, the evasion technique uses emulator environment detection, root detection, debugger detection, and so on. We also needed to respond to android container environment detection. We conducted android container environment detection through vulnerability analysis on A-Pot. As a result, it detected various android container environments. Therefore, currently we have modified the android container configuration part. If an attacker wants to detect the android container environment, it gives the same results as a real phone. Therefore, it is possible to respond to the environmental analysis that has been indicated to be the most significant limitation of the existing research limitation of the existing research.

VII. CONCLUSION

In this paper, we introduce the A-Pot system that is built to deal with intelligent Android malware. The A-Pot system uses Android container technology. In addition, various sensor modules and mini base stations were used to improve the performance of the dynamic analysis. Through this, we overcome the difficulties of intelligent malware analysis. In the past, many studies have provided only the analysis performance results of the malware that can be analyzed. In other words, the results were studied except for intelligent malware, which is difficult to analyze dynamically. However, we were able to get more than 97% of the malicious apps to run include intelligent malware. In existing research, the malicious code analysis rate of Android may be high, but the malicious code analysis rate cannot be accurately determined because it only judges malicious code which does not work. However, we were able to get more than 97% of malicious apps to run. In addition, while running many apps, we confirmed that the analysis time was faster than that of the existing system. Existing methods of analysis using real phones include the process of connecting and analyzing the ADB to the rooted device. In addition, it takes 30 minutes to reconfigure the system after analysis. A-Pot, however, initialized and reconfigured the system in approximately 90 seconds. In addition, A-Pot solves the problem of analysis conducted by operating in the emulator. However, due to the Intel x86-based CPU environment, rather than the general Android CPU environment, NDK analysis is not possible, and the operation rate is very low. However, A-Pot was able to analyze the NDK region, which increased the performance of the analysis. Through this, various analysis results can be provided. However, in our study, it is difficult to determine whether malicious behavior is performed after a specific action such as a logic bomb. To overcome this problem, we plan to conduct a study on the solution to the logic bomb.

REFERENCES

[1] J. Oberheide and C. Miller, "Dissecting the Android bouncer," in *Proc. SummerCon*, New York, NY, USA, vol. 95, 2012, p. 110.

[2] *Google Play Protect, Google Android*. Accessed: May 17, 2019. [Online]. Available: https://www.android.com/intl/en/play-protect/

[3] USPCNET. *Google Continues to Battle With Malware in Play Store*. Accessed: May 17, 2019. [Online]. Available: https://www.uspcnet.com/2018/12/06/google-continues-to-battle-with-malware-in-play-store/

[4] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, 2014, pp. 576–587.

[5] K. Bakour, H. M. Unver, and R. Ghanem, "The Android malware static analysis: Techniques, limitations, and open challenges," in *Proc. 3rd Int. Conf. Comput. Sci. Eng. (UBMK)*, Sep. 2018, pp. 586–593.

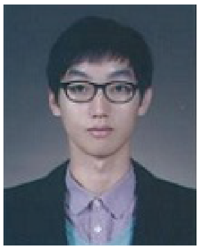
[6] Z. Meng, Y. Xiong, W. Huang, L. Qin, X. Jin, and H. Yan, "AppScalpel: Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications," *Neurocomputing*, vol. 341, pp. 10–25, May 2019.

[7] A. Martin, R. Lara-Cabrera, and D. Camacho, "A new tool for static and dynamic Android malware analysis," in *Data Science and Knowledge Engineering for Sensing Decision Support*. Singapore: World Scientific, 2018, pp. 509–516.

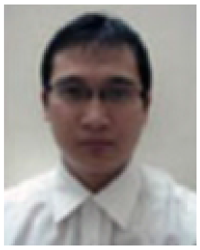
[8] M. Y. Wong and D. Lie, "IntelliDroid: A targeted input generator for the dynamic analysis of Android malware," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 21–24.

[9] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," 2016, *arXiv:1612.04433*. [Online]. Available: http://arxiv.org/abs/1612.04433

[10] Y. Xue, G. Meng, Y. Liu, T. H. Tan, H. Chen, J. Sun, and J. Zhang, "Auditing anti-malware tools by evolving Android malware and dynamic loading technique," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1529–1544, Jul. 2017.
[11] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.
[12] S. Rasthofer, I. Asrar, S. Huber, and E. Bodden, "How current Android malware seeks to evade automated code analysis," in *Proc. IFIP Int. Conf. Inf. Secur. Theory Practice*. Cham, Switzerland: Springer, 2015, pp. 187–202.
[13] V. Afonso, A. Kalysch, T. Müller, D. Oliveir, A. Grégio, and P. L. de Geus, "Lumus: Dynamically uncovering evasive Android applications," in *Proc. Int. Conf. Inf. Secur.* Cham, Switzerland: Springer, 2018, pp. 47–66.
[14] FOSSHUB. *Remix OS*. Accessed: May 17, 2019. [Online]. Available: https://www.fosshub.com/Remix-OS.html
[15] P. Irolla and E. Filiol, "Glassbox: Dynamic analysis platform for malware Android applications on real devices," 2016, *arXiv:1609.04718*. [Online]. Available: http://arxiv.org/abs/1609.04718
[16] M. Kim, T. J. Lee, Y. Shin, and H. Y. Youm, "A study on behavior-based mobile malware analysis system against evasion techniques," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2016, pp. 455–457.
[17] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of Android malware using hybrid techniques," *Digit. Invest.*, vol. 24, pp. 25–33, Mar. 2018.
[18] A. Desnos and L. Patrik, "Droidbox: An Android application sandbox for dynamic analysis," Lund Univ., Lund, Sweden, Tech. Rep., 2012. [Online]. Available: https://code.google.eom/p/droidbox
[19] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," presented at the 21st USENIX Secur. Symp., 2012.
[20] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 1–29, Jun. 2014.
[21] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. V. D. Veen, and C. Platzer, "ANDRUBIS–1,000,000 apps later: A view on current Android malware behaviors," in *Proc. 3rd Int. Workshop Building Anal. Datasets Gathering Exp. Returns Secur. (BADGERS)*, Sep. 2014, pp. 3–17.
[22] B. A. Debelo, W. Pak, and Y.-J. Choi, "Sandroid: Simplistic permission based Android malware detection and classification," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2013, pp. 1683–1687.
[23] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile protection for smartphones," in *Proc. 26th Annu. Comput. Secur. Appl. Conf.*, 2010, pp. 347–356.
[24] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic reconstruction of Android malware behaviors," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.
[25] N.-T. Chau and S. Jung, "Dynamic analysis with Android container: Challenges and opportunities," *Digit. Invest.*, vol. 27, pp. 38–46, Dec. 2018.
[26] ODROID. *HardKernel*. Accessed: May 17, 2019. [Online]. Available: https://www.hardkernel.com/ko/
[27] R. J. Behrouz, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann, "EcoDroid: An approach for energy-based ranking of Android apps," in *Proc. IEEE/ACM 4th Int. Workshop Green Sustain. Softw.*, May 2015, pp. 8–14.
[28] Nuand. *BladeRF*. Accessed: May 17, 2019. [Online]. Available: https://www.nuand.com/product/bladerf-xa4/
[29] *Android Malware Dataset*. Accessed: May 17, 2019. [Online]. Available: http://amd.arguslab.org/
[30] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.



JUNGSOO PARK received the B.S. and M.S. degrees in electronics engineering from Soongsil University, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree in software convergence. His research interests include cloud security, mobile security, and machine learning.



NGOC-TU CHAU received the M.S. degree in information and telecommunications and the Ph.D. degree in computer automation and network from Soongsil University, in 2012 and 2019, respectively. Since 2013, he has been joined the Communication Network Security Laboratory, Soongsil University, where he is currently a Postdoctoral Researcher. Before joining Soongsil University, he worked with Toshiba Software Development (Vietnam) Company Ltd., for three years, as a Senior Software Engineer for two years, a Team Leader for one year, and subsequently as a network administrator position for one year. His research interests include cloud computing and security as a service. Recently, his works related to mobile security, improving Android malware analysis by combining with machine learning.

LONG NGUYEN-VU received the B.S. degree in computer science from the Vietnam National University of Information and Technology, in 2012, and the M.S. degree in electronic engineering from Soongsil University, Seoul, South Korea, in 2016. His research interests include big data, mobile security, cloud security with emphasis on malware analysis, and system design.

JAEHYEON YOON received the B.S. degree in computer engineering from the Soongsil University Lifelong Education Institute, in 2016, and the M.S. degree in software convergence from Soongsil University, in 2018, where he is currently pursuing the Ph.D. degree in electronics engineering. His research interests include cloud security, mobile security, and machine learning.

SOUHWAN JUNG received the B.S. and M.S. degrees in electronics engineering from Seoul National University, in 1985 and 1987, respectively, and the Ph.D. degree from the University of Washington, Seattle, WA, USA, in 1996. From 1996 to 1997, he was a Senior Software Engineer with Stellar One Corporation, Bellevue, WA, USA. In 1997, he joined the School of Electronic Engineering, Soongsil University, Seoul, South Korea, where he currently serves as a Professor. He is also an Executive Director of the Korea Institute of Information Security and Cryptology. He was also a Program Director of the Ministry of Knowledge Economy in Korea for information security area from 2009 to 2011. He has led the Smart Security Service Research Center funded by the Korean Government for six years since 2012. His current research interests include Android and malware security, cloud security, and the IoT security.