IJICTR International Journal of Information & Communication Technology Research ITRC

Volume 9 - Number 2 - Spring 2017(11-17)

An Attack-Defense Model for the Binder on the Android Kernel Level

Majid Salehi
Sharif University of Technology
Tehran, Iran
masalehi@ce.sharif.edu

Mohammad Hesam Tadayon
Iran Telecommunication Research Center (ITRC)
Tehran, Iran
tadayon@itrc.ac.ir

Farid Daryabar
Iran Telecommunication Research Center (ITRC)
Tehran, Iran
f.daryabar@itrc.ac.ir

Received: December 9, 2016 - Accepted: March 2, 2017

Abstract—**In this paper, we consider to seek vulnerabilities and we conduct possible attacks on the crucial and essential parts of Android OSs architecture including the framework and the Android kernel layers. As a regard, we explain the Binder component of Android OS from security point of view. Then, we demonstrate how to penetrate into the Binder and control data exchange mechanism in Android OS by proposing a kernel level attack model based on the hooking method. In addition, we provide a method to detect these kinds of attacks on Android frameworks and the kernel layer. As a result, by implementing the attack model, it is illustrated that the Android processes are detectable and the data can be extracted from any process and system calls. On the other hand, by using our detection proposed method the possibility of using this attack approach in the installed applications on the Android smartphones will be sharply decreased.**

Keywords- smartphone security; android security; android penetration testing; binder component; kernel level attack

I. INTRODUCTION

Nowadays, the technology regarding smartphone devices has shown revolutionary development over the past few years. from year 2009 to 2014, there had been a sharp increase in the rate of smartphone usability in different kind of areas and applications, approximately 88% per year [1], [2]. There are vast varieties of factors that have a great influence on our daily life, but just like the two sides of a coin they offer both benefits and drawbacks, and smartphones are of no exception. Considering the fact that PCs have been around for a long time compared with smartphones, the reported existing malware for smartphones have been essentially and practically simpler than created PCs' malware so far [3].

The usability of PCs is increasingly shifting toward smartphones. Thus, smartphones grew to become subjects to the same or even greater vulnerabilities as PCs. Android malware can violate users' privacy and they can access users' confidential information using different malicious methods [4]. The majority of the written malicious codes for Android OS thus far has targeted the upper layers of Android OSs; however, the lower layers including the framework layer and the kernel layer have not been targeted and penetrated up to this point [3].

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

Moreover, the issue common among all the Android malware is that all of which are designed with a general knowledge in upper layers of Android OSs' architecture. As a result, this issue leads us to consider on a research in detecting vulnerabilities and conducting possible attacks in the crucial and essential part of Android OSs architecture including the framework and the kernel layers in depth.

In this paper, we consider the structure and architecture of one of the most critical and fundamental components in Android OSs that have been studied in [23]. Moreover this significant area has lack of literature by the researchers and practitioners. This component is named Binder which is a vital component and it plays the role of a bridge between upper layers and the lower layers of Android OSs [5]. With implementing an attack on Binder in order to take possession of it, we will have a grant and an executed control of all the data exchanged between applications and services. Actually, the nature of characteristic and features of Binder provide us accessibility to all communicated information and sensitive data on Android. Therefore, taking control of the Binder approaches to take control of the whole Android OS.

As mentioned, Android services and applications need to communicate and share data to facilitate inter-process communication through the Binder component. Binder is implemented in layers of application and kernel [3], [5]. The reference of [3], is the only research that considered the security of this critical component in the application layer; however, there is no consideration on the kernel layer. Moreover, the rest of the existing literatures considered on the architecture of Binder and there is no contribution on the security point of view in this vital component especially the Binder in the kernel layer. Therefore, we consider the analysis and investigation of the Binder architecture in the kernel layer for the first time. We provide an attack model based on Hooking method which can be utilized to capture and extract all the messages exchanged through the Binder driver.

Additionally, to detect and prevent this hooking method at the user and kernel levels, we proposed a detection method which makes these kind of attacks almost impossible.

This paper consists of an introduction and follows the sections which describe the fundamental concepts in Section 2, the related works in Section 3, the proposed attack model in Section 4, the implementation in Section 5, and the evaluation of the proposed model in Section 6. In the end, we conclude the paper and follow up with future research opportunities in Section 7.

II. FUNDAMENTAL CONCEPTS

In this section the fundamental concepts of Android architecture and data exchange mechanism are provided.

A. *Android architecture*

As illustrated in Fig. 1, Android OS is a layered OS and its components are divided into three layers. The lowest layer is the kernel layer which provides Linux kernel to the upper layers. The duty of this layer is management of the network services, drivers, file systems, memory, and process. Therefore, Android OS is designed base on the Linux kernel.

The second layer is named Middleware which is divided into three parts. The first part is C and C++ native libraries that are included libc, SQLite, FreeType, SGL, SSL, WebKit, surface management, media framework and etc. The second part is the application framework that provides APIs with different functionalities and services such as setting an alarm or reminder, accessing the location information, phone calls, and etc. The application framework is divided into two important parts. The first part is activity manager that controls and monitors the requested access permissions to different services. The second part is package manager that is responsible for installing and managing permissions. The third part of Middleware is named Android runtime that is included kernel libraries and Dalvik Virtual Machine (DVM). DVM ensures that applications run in systems with relatively smaller RAM, slower processors and without swap space. The third layer is the application layer. It is written in java and it provides a connection between end users and applications. The provided applications in this layer run in its own DVM and they can read native codes from native libraries using JNI [3], [5].

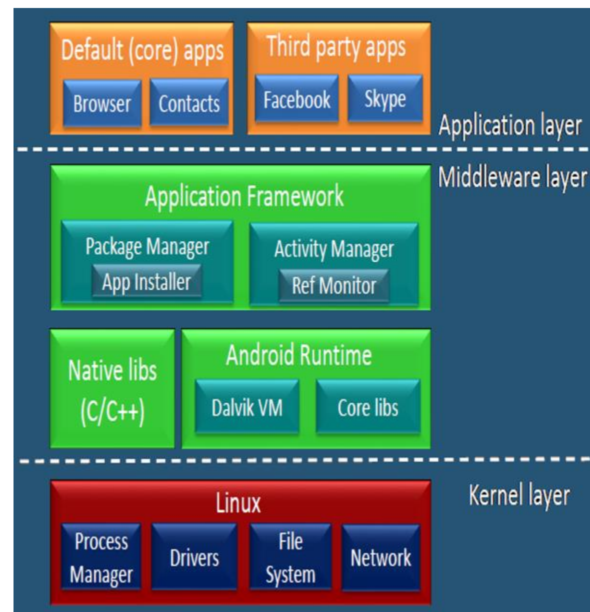

Figure 1. Android's architecture, the classic diagram.

B. *Data Exchange Mechanism*

In the android architecture, all the applications can be run only in their specific process area meaning that they can only access to their files and data. The reason of providing this security architecture is to protect the applications and their sensitive data from the existing malware. However, the applications and services need to communicate with each other. Therefore, there is a data exchange mechanism for that reason. Regarding

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

this mechanism, Intents are playing the role of communicators between activities, services and applications. Each Intent is a message that contains all the information and data that need to be conveyed, including the receiver information and the data. A Binder driver is implemented at kernel layer. Hence, all the transferred data between processes and applications pass through this driver. In addition, all the intents and the exchanged messages in the processes and processes components pass through the Binder.

The Binder component is implemented at two layers. The first layer, that is named "The Binder Framework", is a user-level library called *libbinder*. This is loaded into most processes on Android OSs. The task of this library is wrapping the requests and send them via system calls. On the other hand, un-wrapping objects and creating the respond objects in service processes are done through this library. Those objects that are created by Binder are called *Parcel [5]*.The second layer is named, The Binder driver that controls all the process communications in kernel level. In fact, the Binder library at the user level by calling system functions sends the required messages and requests to this driver, then the Binder driver drives them to the specific service or process.

III. RELATED WORKS

An analysis of different malware techniques and their countermeasures was conducted in [6]. The authors proposed a novel method for malware development and attack techniques in the area of mobile botnets, usage pattern based attacks and repackaging attacks. It takes the read contacts permissions, send SMS permissions and their malware sends an auto response to miscalls.

The reference of [6], [7] mentioned one of the new malware that take advantage of users and deceit them using a technique called repackaging. This technique is highly effective because users have difficulties to find out the difference between a legitimate application and the malware. In this technique, the malware tries to reverse engineer popular and legitimate applications, modifies them to inject malicious code after that republish them to the market. As a result, because of the popularity of the applications, users download them without having a knowledge that the applications are taking advantages of their information and smartphones. Among the researches that has been conducted on detection techniques of these kind of malware, the research of [8] can be mentioned. The authors, presented DNADroid Android malware detection tool. The proposed tool used a technique based on program dependence graphs (PDGs) to obtain the similarities between the malicious and legitimated applications and detect the repackaging technique.

In reference of [9] a tool named Applink was extended to detect the repackaging technique using watermarking technique. Additionally, the authors of [10] extended DroidMOSS tool by using hashing algorithms to extract the similarities between the repackaged malware and eliminated applications. In the research of [11], a fingerprint method in the layer of applications' code was used to detect and analyze the repackaging techniques. Moreover, JuxtApp is a tool that extracts static features of codes, then by showing the bit vectors, it compares the application to detect the malicious activity [12].

By using and taking advantage of Firmware, some other methods of Android malware expand and infect smart systems these days. In this technique, there are some applications that are pre-installed by the Firmware creator. The users cannot uninstall these application unless they have root access to their android device. The pre-installed applications have privileged access comparing to other application in the smartphones.

In [13], Droidray tool is used to evaluate the Firmware using static and dynamic methods, and then they store the result in their own database and illustrate them in an organized form. The reference of [14] analyzed ten kind of Firmware and they investigate the installed applications' permissions. Additionally, they investigated the vulnerabilities that can be lead to information leakage or illegal access to the system resources by the Firmware. As a result, 85.87% of the pre-installed applications on the Firmware requested more that required permission of the systems. Also 64.71% to 85% of the vulnerabilities in the firmware are because the personalization that companies impose on them.

On the other hand, Android operating system access control model was considered by many researches. One these researches is [15] that Android's internal components and their relationships were analyzed. Based on the research of [16], most of the Android applications request more that required permission from users during the installation. Therefore, this is a critical topic that is analyzed and considered by the researches of [17]–[19]. The Coarse Grained permissions in android is another weaknesses of Android system. Based on this weakness the provided permissions to applications allow them to access multiple APIs that are unnecessarily [16]. However, the reference of [20] proposed a tool for Fine Grained APIs.

Regarding the Android system access control model, users are not able to give permissions to the applications in different circumstances. Based on this issue, the researches of [21], [22] focused on method that provides users the ability of giving permissions from context.

The remarkable common issue to all previous researches on Android malware indicates the exiting malware still have long way to go and they are created to be executed on the upper layers of the Android OSs; hence, they are easily detectable. Last but not least, the only research that considered on introducing methods to design Android malware in the Android kernel level based on the exchanging data mechanism is the reference of [3]. In this research, the authors used a code injection method in the layer of framework to track the communication data in Binder. Nevertheless, this

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

method is detectable by the simple detection techniques.

IV. THE ATTACK MODEL

As explained in previous sections, large amounts of information and data exchanged between processes pass through the Binder component. Therefore, controlling this component means controlling the Android OS and the users' smartphones. Based on the existing literature, one of the most vital issues that many Android programmers as well as many security researchers have not considered is that all internal messages and intents of each process, that are exchanged between internal components of a program, pass through the Binder components.

For instance, an android programmer utilizes HTTPS protocol to provide a secure media and communicate with a web-base server, so all the related data transfer through this media. However, before transferring the data, the data are being passed to the network management service through the Binder components unencrypted and in plain text. Then data will be encrypted in the service. Hence, there is a possibility to access those sensitive and unencrypted data through the Binder component.

Based on the explanation, a Hooking method can be used in order to capture all the messages exchanged through the Binder. The followings explain the methods and its different types in detail.

Hooking is a notion of obtaining control of application execution flow without any change and recompile the source code. This is achieved by stopping the function calls and redirecting them to tailor made codes. By injecting the custom code, any operation can be performed. After that, the main function capabilities can be executed and the result can be returned simply or it can be changed to be returned to the code that recalled the Hooked function. The hooking methods are conducted in two levels as follow:

1) *Hooking at the user level:* in this method, a code is injected to the related library. Using this code, a small number of the commands in target system function (the function that we want to be tracked) is replaced by an unconditional jump to the diversion that is created by the attacker. Those Target function commands are stored in a temporary function, which are included commands that have been removed from the target function and it created an unconditional jump to the rest of the target functions. This type of Hooking is the simplest and most widely used approach to intercept the functions.

It is noteworthy that anti-malware programs with a simple hashing functions can detect and prevent the injection codes to services and sensitive processes of Android OS. According to the code injection that is done via a particular system functions, anti-malware programs can track these functions and prevent the code injection.

2) *Hooking at the kernel level:* typically, to organize and instant access to the system functions, Android OS uses an interface table named "system call table". This table contains the addresses of most system functions in the Android OS. When a system function is called within an application by using this table, first application control is returned to the Android OS. Right after that, the OS refers to the system call table and depending on the type of the requested system function and the arguments, the address of the required system function is find and it is called.

Consequently, in order to intercept the system functions, we need to replace the existing addresses in the system call table with our own function addresses, and after utilizing our own function we jump to the original function address.

Considering that this method is done in kernel level, anti-malware applications are not able to detect them easily, and the prevention is much more difficult than the previous method.

Due to the fact that the Binder library uses a system function called *ioctl* to connect with the Binder driver and transfers data, with intercepting ioctl system function it is possible to get access and extract all the exchanged data and information using the Hooking method in Kernel layer.

V. COUNTERMEASURE

According to the previous explanation in the attack model section, it is evident that the main and fundamental methods used in these type of malware is the hooking method. Therefore, in our prevention proposed method, possibility of using this attack approach in the installed applications on the Android smartphones goes away. A detailed explanation of the hooking attack restriction and prevention methods is provided in the following subsections.

A. Hooking prevention at the user level

In order to prevent and deal with this hooking method, a hashing method is utilized to investigate code injections to the critical android OS's services and processes. Considering the fact that this method of hooking attempts to change some part of the application codes, it changes the hashing value of the application or the service as well. Thus, it is obvious to prevent and detect the attack with storing the existing services and applications' hash values on the users' system.

B. Hooking prevention at the kernel level

Fig. 2 illustrates the proposed solution architecture to prevent and deal with the hooking attack on kernel level. As explained the hooking attack method at the kernel level in section 5, the attack is successfully done when the attacker is able to intercept and change the existing addresses in the system call table. Hence, to avoid and deal with these kind of attacks, it is necessary to prevent changing the addresses in the system call table. For this purpose, with developing a kernel module which is periodically check the integrity of the addresses in the system call table, occurring the hooking techniques by malware can be prevented. Details of how the table is going to be checked is set out in the prevention implementation section.

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

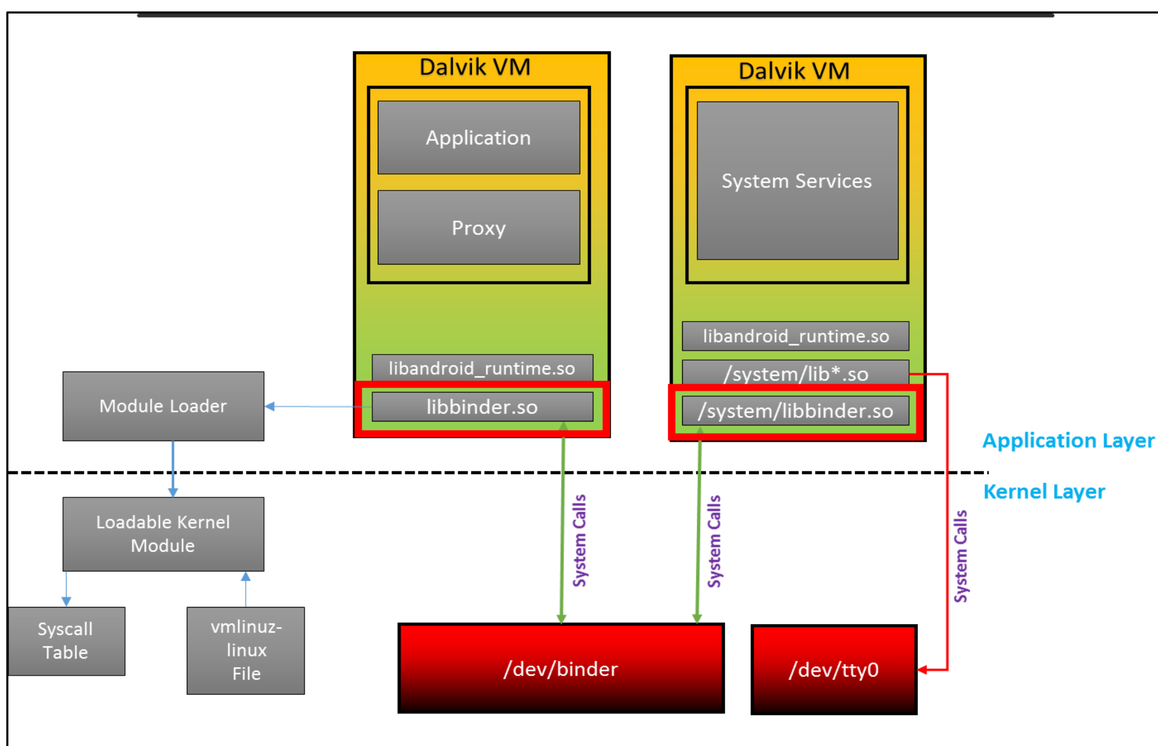

Figure 2. Proposed architecture the hooking attack prevention on the kernel level

VI. Implementation

Implementation of the attack model and detection method are explained in the following subsections.

A. Attack Implementation

In order to extract the data exchanged between processes and applications in Android OS, we designed a kernel module for the Android OS kernel. This module changes the system call table and it modifies ioctl system function address.

Since there are millions of calls per minute in ioctl system function on Android OSs, processing this size of information in the OS kernel level is almost impossible. It is because of the real-time processing. Thus, in order to analyze, intercept and extract the exchanged data between two specific processes accurately and more quickly, we can filter the incoming messages by examining the UID process of the function which has called ioctl system function and the UID of the called services.

Fig. 3 illustrates the structure related to the ioctl system call and the data structure within the ioctl [5]. As shown, the first argument of ioctl system call is the name of the driver that is supposed to receive the data in the form of this system call. The second argument is the request code that is supposed to be given to the drive. And the third argument is an address to the data structure of *binder_write_read* which contains the information and sent commands to the defined service or component. As illustrated the submitted information are sent marshalled.

All data are sent one after another in relevant format. In order to obtain this information and determining the data, it is necessary to unmarshall them. Moreover, with checking the InterfaceToken and code fields, the service and the function will be specified. Then, considering the signature of the service function, the data from the Hooked system call will be extracted. For instant, as illustrated in Fig. 3, the receiver is Isms and the function is sendText.

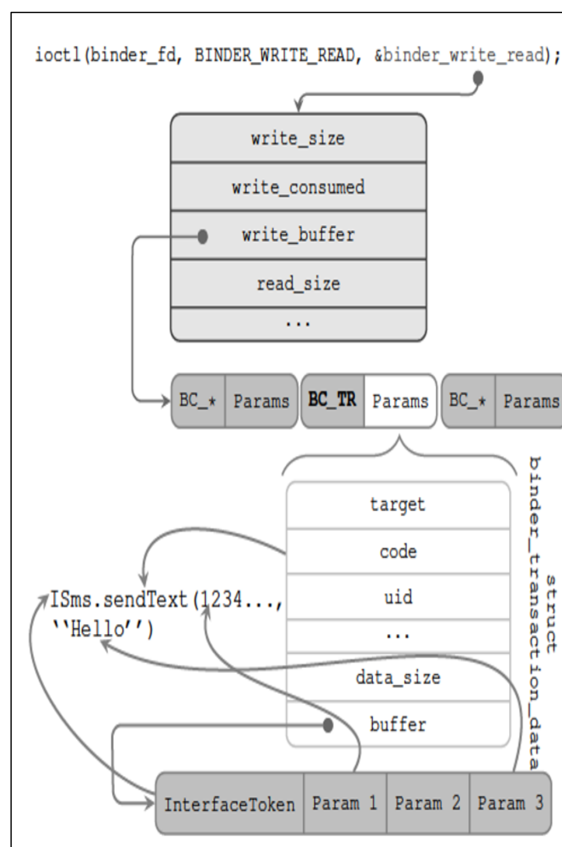

Figure 3 A Binder Payload for SMS process [5].

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

At the end we need to consider that the function address we replace with the original function address of ioctl in the system call table, should have a signature similar to the original one meaning that the input and output parameters must be defined exactly like the original function.

B. *Detection Implementation*

Implementation explanation of the hooking attack prevention methods is provided in the following subsections.

Implementation of hooking prevention at the user level: Hooking for intercepting Binder transactions at user level performed through "*Libbinder.so*" library as shown in Fig 4. Before making any changes in this library, it is necessary to get a hash value of the related file and in different periods of time this hash value be compared with new hash values of the file. Therefore, if any difference is detected during the hash value comparison, the attack is detected and the users can be warned.

Implementation of hooking prevention at the kernel level: In order to correct those addresses that are exist in the memory in the system call table, it is required to find out the actual and primitive addresses of the system functions exist in the table. On the Android OS the system function addresses exist in a file named "*Vmlinuz-linux*". In fact, "*Vmlinuz-linux*" file contains static parts of the Linux kernel such as system calls, which are loaded on the memory during the OS booting operation. Adding these addresses with the address of the "*Vmlinuz-linux*" file in the memory gives the actual address of the system functions in the memory. In this method, firstly the binder library at the user level loads a kernel module. Then, the loaded kernel module performs the system call table correction in case of any changes has occurred.

VII. Evaluation

In order to evaluate the proposed model, we designed and implemented a kernel module for the Android OS Goldfish kernel with version 2.6.29. As specified in the following code, that is hooked in a system function, it is necessary to extract the required data initially. Then we call the main system function.

```
int hooked_open ( const char *pathname, int flags)
{
Before_transaction ( buff );
int ret = open (fd , command, buff );
After_transaction ( buff );
return ret ; }
```

To verify the implemented kernel module, we hooked three system functions called Open, Read and Write, and we capture the logs from their calls by applications or Android OS services. As illustrated in Fig. 5, this module is properly implemented and the system functions are being hooked. In detail, when a system function is called, first the system function of *our_sys_read, our_sys_open,* or *our_sys_write* is executed then the Handle is returned to the original system call.

As shown in Fig. 3, the UID of the process which has called this system function is extractable. For instance, three processes with the UIDs of 1000 and 2987 are visible in the Figure. Consequently, it is possible to hook and analyze the only system function which was called by a process with a unique ID.

```
<...>-321  [000] 30100.362579: our_sys_open: 1000:Our_Sys_Open
<...>-321  [000] 30100.362640: our_sys_read: 1000:Our_Sys_Read
<...>-321  [000] 30100.362762: our_sys_open: 1000:Our_Sys_Open
<...>-321  [000] 30100.362793: our_sys_read: 1000:Our_Sys_Read
<...>-321  [000] 30100.362946: our_sys_open: 1000:Our_Sys_Open
<...>-321  [000] 30100.362976: our_sys_read: 1000:Our_Sys_Read
<...>-201  [000] 30100.363586: our_sys_read: 2987:Our_Sys_Read
<...>-313  [000] 30100.368683: our_sys_read: 1000:Our_Sys_Read
<...>-321  [000] 30100.368835: our_sys_write: 1000:Our_Sys_Write
<...>-313  [000] 30100.369568: our_sys_write: 1000:Our_Sys_Write
```

Figure 4. Hooking the system function.

Considering that the proposed method is designed in a kernel module form and it hooks the system functions in real time mode and extract the information, its executive overhead is equal to calling a normal system function and it is very meager.

VIII. Conclusion

In this paper, from a security perspective, we described the Binder component on Android OS then we investigated its security architecture. Furthermore, with designing an active malware in OS Kernel, we demonstrated how to penetrate into the Binder and control data exchange mechanism in Android OS. By considering the fact that most android malware that have been designed so far, are operating in the higher levels of Android OS. Hence, the detection and confrontation with them can be easily conducted. Besides, the only mechanisms that the malware use to protect themselves are included obfuscation, encryption, social engineering, and etc. These mechanisms are easily detectable. Consequently, it is the time for android malware to be more advanced and be equipped with the knowledge of lower levels of android OS.

As explained, this method is done in kernel level and anti-malware applications are not able to detect them easily, and the prevention is much more difficult than the previous and existing methods. Therefore, we proposed a detection method for these kind of attacks at the user and kernel levels. As a result, using the detection method, the possibility of conducting those kind of attacks will go away.

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

ACKNOWLEDGMENT

This research has been supported by Iran Telecommunication Research Center (ITRC), and we are thankful to them for providing us the vast range of materials besides their encouragements and support to conduct this research.

REFERENCES

[1] N. Samet, A. Ben Letaifa, M. Hamdi, and S. Tabbane, “Forensic investigation in Mobile Cloud environment,” 2014, pp. 1–5.

[2] F. Daryabar, A. Dehghantanha, B. Eterovic-Soric, and K.-K. R. Choo, “Forensic investigation of OneDrive, Box, GoogleDrive and Dropbox applications on Android and iOS devices,” *Aust. J. Forensic Sci.*, pp. 1–28, 2016.

[3] N. Artenstein and I. Revivo, “Man in the binder: He who controls ipc, controls the droid,” *Eur. BlackHat Conf*, 2014.

[4] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 3–14.

[5] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, “CopperDroid: Automatic Reconstruction of Android Malware Behaviors.,” in *NDSS*, 2015.

[6] R. Raveendranath, V. Rajamani, A. J. Babu, and S. K. Datta, “Android malware attacks and countermeasures: Current and future directions,” in *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on*, 2014, pp. 137–143.

[7] I. Lookout, “Lookout Mobile Threat Report August 2011,” 2011.

[8] J. Crussell, C. Gibler, and H. Chen, “Attack of the clones: Detecting cloned applications on android markets,” in *Computer Security–ESORICS 2012*, Springer, 2012, pp. 37–54.

[9] W. Zhou, X. Zhang, and X. Jiang, “AppInk: watermarking android apps for repackaging deterrence,” in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, 2013, pp. 1–12.

[10] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, “Detecting repackaged smartphone applications in third-party android marketplaces,” in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, 2012, pp. 317–326.

[11] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, “Plagiarizing smartphone applications: attack strategies and defense techniques,” in *Engineering Secure Software and Systems*, Springer, 2012, pp. 106–120.

[12] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, “Juxtapp: A scalable system for detecting code reuse among android applications,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2012, pp. 62–81.

[13] M. Zheng, M. Sun, and J. Lui, “DroidRay: a security evaluation system for customized android firmwares,” in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 471–482.

[14] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, “The impact of vendor customizations on android security,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 623–634.

[15] W. Enck, M. Ongtang, and P. McDaniel, “Understanding android security,” *IEEE Secur. Priv.*, no. 1, pp. 50–57, 2009.

[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, “Android permissions demystified,” in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

[17] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, “Pscout: analyzing the android permission specification,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 217–228.

[18] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, “A Study of Android Application Security.,” *USENIX Secur. Symp.*, vol. 2, p. 2, 2011.

[19] S. Jana and V. Shmatikov, “Memento: Learning secrets from process footprints,” in *Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, pp. 143–157.

[20] J. Jeon *et al.*, “Dr. Android and Mr. Hide: fine-grained permissions in android applications,” in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, 2012, pp. 3–14.

[21] M. Conti, V. T. N. Nguyen, and B. Crispo, “CRePE: context-related policy enforcement for android,” in *Information Security*, Springer, 2010, pp. 331–345.

[22] S. Bugiel, S. Heuser, and A.-R. Sadeghi, “Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies.,” in *Usenix security*, 2013, pp. 131–146.

[23] M. Salehi, F. Daryabar, and M.H. Tadayon, “Welcome to Binder: A kernel level attack model for the Binder in Android operating system.,” in 8th International Symposium on Telecommunications (IST), 2016.

Majid Salehi received his B.Sc. degree in computer engineering from Isfahan University, Isfahan, Iran in 2010, and his M.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran in 2016. He is currently a researcher with the DNS Laboratory at Sharif University of Technology. His research interests include Malware detection, OS security, and information forensics.

Mohammad Hesam Tadayon received his M.Sc. degree in mathematics from the University of Tarbiat Modares,Tehran, Iran, in 1997, and his Ph.D. degree in applied mathematics (coding and cryptography) from the University of Tarbiat Moallem of Tehran (Kharazmi), Tehran, Iran, in 2008. He has been holding an Assistant Professorship position with Iran Telecommunication Research Center (ITRC) since 2008. He is a member of national councils in the Iranian Ministry of Science and Technology. He has served in many research and industrial projects. His research interests include information theory, error-control coding and data security.

Farid Daryabar is a cybersecurity researcher-developer with Iran telecommunication Research Center. He graduated from the University Putra Malaysia with a Master of Science (Cybersecurity/Forensic). He has (co)authored several publications in Cybersecurity area. Farid has awarded a silver and two bronze medals in R&D Invention/Innovation (PRPI12 and MTE13), CEH and CHFI.

[Downloaded from journal.itrc.ac.ir on 2024-11-01]

IJICTR

This Page intentionally left blank.

[Downloaded from journal.itrc.ac.ir on 2024-11-01]