International Journal of Computer Networks and Communications Security

VOL. 2, NO. 9, SEPTEMBER 2014, 318–327
Available online at: www.ijcncs.org
ISSN 2308-9830

Firewall-based Solution for Preventing Privilege Escalation Attacks in Android

Ali Magdy[1], Mohsen Mahros[2], Elsayed Hemayed[3]

[1] Bank Misr, Department of Information Comm. Technology, Cairo, Egypt

[2, 3] Cairo University, Department of Comm. and Electronics, Cairo, Egypt

E-mail: [1]alymagdy2013@yahoo.com, [2]mohsenmahroos@msn.com, [3]hemayed@ieee.org

ABSTRACT

In this paper, we are proposing a Firewall-based solution for protecting Android operating systems against privilege escalation attacks, mainly, confused deputy attacks and collusion attacks. The proposed Firewall protects the applications that have critical privilege permission. Any other applications without the critical permission will not be able to call protected applications via privilege escalation attacks. Since the Internet is the door of attack, we consider the permission to access Internet as a critical permission. As such, any application cannot access the Internet directly or indirectly, through privilege escalation, without confirmation of the user disallowing invulnerable leakage of private data. The proposed solution allows also protection to different critical permissions through the creation of multi-critical protection zones. We implemented the multi-critical protection zones by selecting READ_CONTACTS permission and INTERNET permission as critical permissions and the applications having one of these permission or both, they will be protected by our firewall against the privilege escalation attacks. The efficiency and effectiveness of the proposed solution are evaluated in this paper along with the imposed overhead. The evaluation includes the Android with one zone firewall and with two zones firewall.

Keywords: *Privilege escalation attacks, Android Security, Collusion Attack, Confused Deputy Attack, Excessive privilege Attack.*

1 INTRODUCTION

The Android operating system is one of the most important operating system; its importance is due to the wide spread in recent years as an operating system for Smart phones and tablet computers. The wide spread of Android leads to increasing attempts of penetration and exploitation of its security weaknesses. Since Android is an open source code, it is easier for hackers to study the system and find its security holes and exploit its weakness.

The behavior of the Android security system is based on identifying the granted permissions and privileges that are required by the Android applications at installation time. Then those granted permissions and privileges are used by Discretionary Access Control (DAC) to secure and isolate the applications from each other and also from system resources. However, DAC mechanism cannot protect the system against malicious applications that may have the ability to indirectly access and cannot also protect the application's components that are not protected by a protection permission. The indirectly access means that an application cannot access a component of another application because this application was not granted the privilege to access this component and succeeded to access this component by exploiting another application in accessing to this target component.

Due to the defects and weaknesses of the DAC mechanism, other attempts were introduced to overcome these weaknesses. For example, SE Android [1] and [2] uses a Mandatory Access Control (MAC) mechanism beside DAC in the kernel layer. SE Android enforces a system-wide security policy over all operations. So it is not enough for an application to own the needed permission and privilege to achieve the required action but also this action must match the system-

wide security policy. Even though SE Android contributes in mitigating privilege escalation attacks, it is still possible for the malicious Android applications to achieve privilege escalation attacks by sending its message indirectly by exploiting intermediates applications or by exploiting directly another application that owns more privileges and that is proved in this paper.

Privilege escalation attacks have three main types, confused deputy, collusion attack and excessive privileges. The confused deputy attack occurs (see Figure 1) when a malware application does not have the required permission to download files for example but it calls a browser to download these files on its behalf.

The collusion attack occurs (see Figure 2) when two or more applications work together to bypass the permissions system [3]. Thus if an application does not have the required permission "P1" to call a component of another application protected by a permission "P1" then this application can successfully call this component by exploiting another application with the required permission "P1".

The excessive permissions attack occurs when the application has more than the required access permissions and privileges. Namely, an application may have access to hardware components such as microphone and camera so it updates its rights to receive and transmit locations and SMS messages without informing the user since this application is seen as having these permissions. Attackers may also exploit this application to make unwanted action.

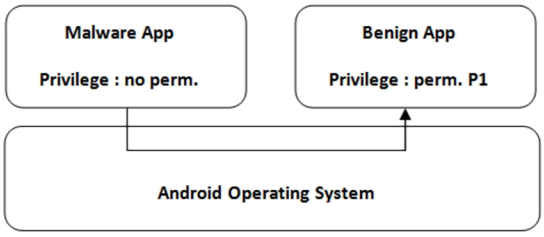


Fig. 1. Confused deputy attack

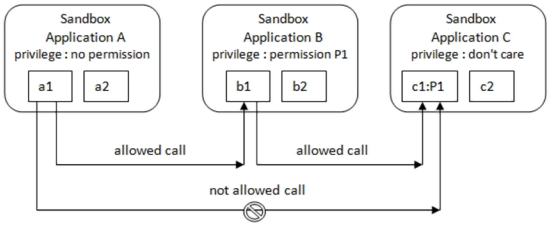


Fig . 2. Collusion attack

In this paper, we are proposing a firewall-based solution for protecting the latest version of Android (Android 4.4) against privilege escalation attacks. The proposed solution is based on the privilege permission itself. An application with a critical permission is considered in a critical zone protected by a firewall. Other applications cannot communicate with the protected application unless they have the same critical permission. Moreover, the proposed firewall can protect multiple zones, each with different critical permission. We implemented the multiple zones firewall considering READ_CONTACTS permission and INTERNET permission as critical permissions and evaluate the effectiveness of this firewall in preventing the privilege escalation attacks by protecting the applications own one or both of these selected permissions in additional to proving no negative effect on the performance of Android operating system.

The remainder of this paper is as follows. We discuss Android Architecture related to our added Firewall in section 2. Related works are discussed in section 3. In Section 4, we show our proposed protection scheme. In Section 5, we evaluate the performance and effectiveness of SE Android before and after adding our Firewall mechanism. Finally the conclusion and future work are presented in Section 6.

2 ANDROID ARCHITECTURE

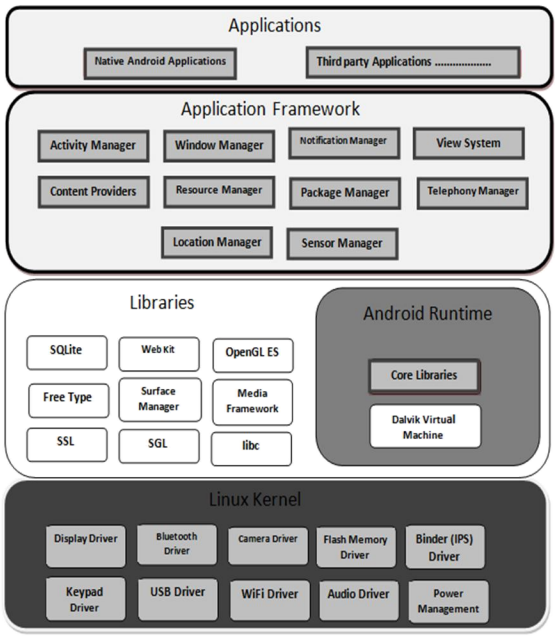


Fig. 3. Android Architecture

The Android platform (see Figure 3), consists of four layers: Application layer, Application Framework layer, Libraries and Run time layer, and Linux Kernel layer [4] and [5]. Our Firewall is located in the Application Framework layer mainly in ActivityManager that interacts with the overall running applications in the system and it can allow or abort the calls or the messages between running applications.

In ActivityManager, the system checks the required permission of the target component if it exists or not in the Manifest file of the sender application. The ActivityManger allows the call if the required permission is Null or exist in the Manifest file of the sender application. It also allows the call if it is between components in the same application or the user is a root user. In this paper, we added a new feature to Android system where the call between applications is checked and permitted or aborted by our Firewall if this call is directed to an application protected by our Firewall.

3 RELATED WORK

In this section, we discuss in brief previous work related to privilege escalation attack mitigation. Saint stands for Secure Application INTeraction. Saint [6] and [7] creates access control policies to protect the application components. Saint enforces policy on installation time and running time. On install-time, the policy is to manage granting the permissions to applications may be installed on the device. For example, if an application declares that it needs a permission P1 and this permission P1 is considered as one of the dangerous permissions then the system will demand confirmation by the user to grant this required permission to the application. But in Saint mechanism, the system does not grant this permission to any application unless the policy of Saint is satisfied with granting this permission P1 to the application beside the confirmation by the user. Next, this application can be installed on the device. At run-time, if the caller application owns the required privilege to access a component of the target application, the system will allow this call. But in Saint mechanism, the system owns additional policy located in the middleware framework to manage the communications and the interactions between the components of Android applications consequently no call, inter-process communication (IPC), or Intent message executed successfully unless this communication satisfying the run-time policy enforced by Saint. In Saint mechanism, if an application wants to make an access call to a component in another application, then this application should have the required permission from the desired component but if this application does not have the required permission, it can exploit another application that has permission to access the desired component. Thus, Saint's mechanism is capable of protecting against confused deputy attack and handles excessive permissions attacks but cannot protect against collusion attack.

MockDroid [8] and [6] prevents the applications from accessing critical information and important resources. When an application requests accessing to critical information, MockDroid presents to this application fake or empty information instead of the requested critical information. MockDroid enables the user to redirect the application to access this critical information at run-time or to deny that access. Thus MockDroid only handle excessive permissions attacks and not the deputy attack nor the collusion attack.

XManDroid stands for eXtended Monitoring on Android. XManDroid [4] and [6] mitigates the privilege escalation attacks by checking the inter component communications among the applications. XManDroid verifies that these inter component communications "ICCs" are matching with the extended system policy and aborts every call that does not achieve those extended system policy. XManDroid can mitigate the collusion attacks and the confused deputy attacks but not the excessive permissions attack.

Kirin's mechanism [9], [6], and [10] works at installation time where Kirin is introduced for providing lightweight certification at install time in order to make inspection for each application installed on the Android device. Kirin faces the malware applications during the installation time by enforcing certain rules that prevent third-party applications from acquiring privileges enabling them to make harmful action on the device or the user. So Kirin can handle the excessive permission attacks.

SE Android [1] approach is similar to XManDroid where Android uses SE Linux in enforcing mandatory access control (MAC), used in the Linux Kernel, beside discretionary access control (DAC). It is not enough for the caller application to own the required permission to access the desired component but also the call should realize the system-wide security policy. Thus SE Android can mitigate the collusion attacks and the confused deputy attacks.

IPC Inspection defines the confused deputy attack that it is a permission re-delegation [11]. Permission re-delegation occurs if it is not authorized for application to perform a certain task, and it can do that via another authorized

application. IPC Inspection is a mechanism to handle the permission re-delegation by preventing any application from escalating its privilege to do unauthorized action. IPC Inspection aims to prevent the permission re-delegation as follows. When an application receives an Intent message from another application, IPC Inspection reduces the privileges of the callee application to look like the privileges of the caller application. IPC Inspection demands that each application wants to call another, they should own the same privilege. This is from our point of view impractical and restricts communication among the applications within the operating system in the form that can negatively affect the performance efficiency of the operating system. IPC Inspection handles only the confused deputy attacks, and the IPC Inspection's authors do not talk about the collusion attacks.

TrustDroid is a security framework for making domain isolation on Android in order to address unauthorized data access, and to secure interaction among the applications of different trust levels [12]. TrustDroid isolates the applications based on assigning each application into a trust level, and makes the applications that are in the same trust level with one color different from the other trust levels. TrustDroid divided the applications into three trust levels: First, pre-installed system applications (native applications). Second, trusted third party applications introduced by the enterprise. Third, un-trusted third-party applications downloaded from public sources, such as Android Market, i.e., Google play. Trusted applications and un-trusted applications have to be isolated from each other and prevented from communication with each other. All installed applications must be able to access the system applications. The Mandatory access control (MAC) of TrustDroid detects inter-component communication (ICC) when ICC is between a caller application and callee application with different colors and prevents this communication.

4 PROPOSED PROTECTION SCHEME

The key policy of the proposed firewall is to make the users of Android devices aware of what is happening in their devices without headache. Our proposal depends on selecting a critical privilege permission of the dangerous privilege permissions then setting all applications with this permission in a zone protected by a firewall. Then calls are allowed without any restriction of firewall between applications in the same zone. Application outside the firewall will not have access to any of the protected applications unless it has the selected permission and as such it has to be a member of this zone. Applications outside the firewall can communicate together without passing on our firewall. The general basic concept of firewall, the high risk levels can access the lower levels but the lower levels cannot access the higher levels so the applications inside the firewall can access any application outside the firewall and not vice versa. The applications in the different zones cannot communicate together.

4.1 Proposed Protection Scheme Methodology

There are many types of firewalls; packet filters, application firewalls, proxy firewalls, and network address translation [13] and [14]. Our proposed firewall can be considered as an application firewall complemented by mandatory access control (MAC). In application firewall, filtering rules are applied to allow or block on a per process basis. Our proposed firewall resides in the ActivityManager enforcing access control rules on all applications (see Figure 4).

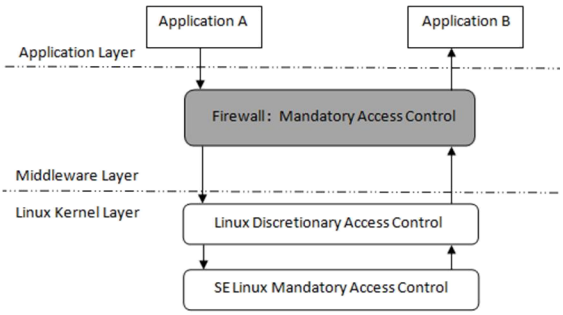


Fig. 4. Proposed protection scheme

The flow chart of allowing or dropping the call is shown in Figure 5. The additional part that is enforced by our firewall is represented by the dotted gray shape in step 3. Our proposed firewall, as shown in Figure 6, has two stages; getting information and taking decision. In first stage, information is gathered about the permissions exist in the manifest file of the source and target applications. In second stage, decision is made to allow or block the call according to the information gathered in the first stage. Calls are blocked by firewall if the critical permission exists in the target and not in the source app.

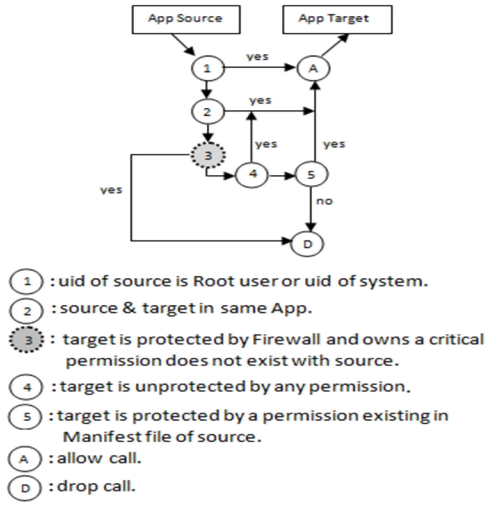


Fig. 5. Allowing and dropping calls flowchart

.Considering the permission to access Internet as one of the most critical permission in our case (leaks of private user data or downloading malicious files are done through an application owns INTERNET permission), Figure 7 shows an example of our proposed solution in case of selecting INTERNET permission as critical permission. As shown in this example, no application can access the Internet without confirmation of the user and as such any application that did not declare this need during installation time will not be able to access Internet by any way.

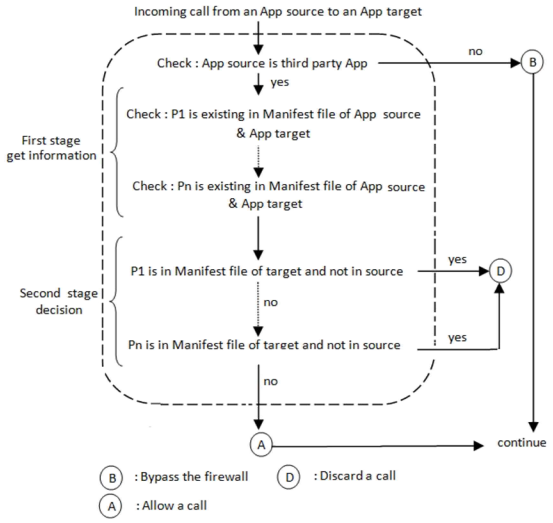


Fig. 6. Proposed firewall stages

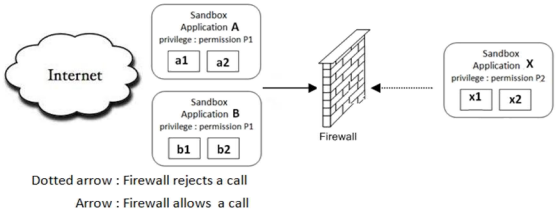


Fig. 7. Firewall protection of Internet access

.The proposed firewall can be also used to protect multiple critical permissions simultaneously through the creation of different zones preventing applications without a certain critical permission from escalating its privilege to this critical permission. As shown in Figure 8, applications in the first zone cannot access applications in the second zone and vice versa. Also any other applications outside the Firewall cannot access neither applications in the first zone nor in the second zone.

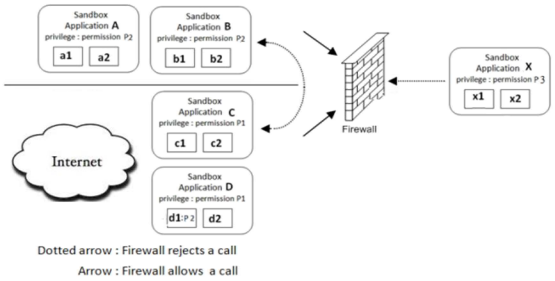
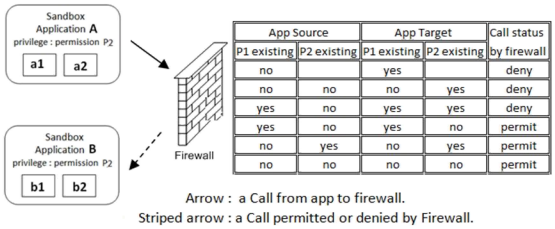


Fig. 8. Firewall with two zones

.In Figure 9, we illustrate the firewall decisions in case of two critical privilege permissions. As shown in the figure, calls will not be permitted if the app target has more critical permissions than the app source.



App Source		App Target		Call status
P1 existing	P2 existing	P1 existing	P2 existing	by firewall
no		yes		deny
no	no	no	yes	deny
yes	no	yes	yes	deny
yes	no	yes	no	permit
no	yes	no	yes	permit
no	no	no	no	permit

Fig. 9. Firewall with two critical permissions

.

4.2 Implementation

Our implementation is based on the source code obtained from the Android Open Source Project (AOSP) [15] and the Android package interfaces (APIs) [16]. Based on the documentation of the AOSP and the XManDroid, we decided to add our java code representing the Firewall in the ActivityManager. In our implementation, we modified the ActivityManager to send log messages containing the UIDs of the source and the receiver applications, which permission required to pass the call, and which critical privilege permissions exist in the source and the receiver applications. Depending on monitoring these log messages, we studied how to make our firewall. Namely, based on the UIDs, the firewall determines if this is a third party application or not and based on the permissions, the firewall allows or rejects the calls.

4.3 Comparison between our Proposal and Some Previous Mechanisms

In this section, we target making comparison between the previous mechanism which may be believed that they are similar to our proposal.

IPC Inspection is to reduce the privileges of the callee application to the intersection privileges of the caller application during the communication between them otherwise each application maintains its privileges permissions. IPC Inspection just handles the confused deputy attacks.

TrustDroid divided the applications into three trust levels: pre-installed system applications (native applications), trusted third party apps introduced by the enterprise, un-trusted third-party applications downloaded from public sources. All installed applications should be able usually to access the system applications. The un-trusted applications are prevented to communicate with the trusted applications. TrustDroid sets one color to the applications at the same trust level and this color is different from the colors of other trust levels. TrustDroid restricts communication between the applications that have different color and it permits for applications that have the same color to communicate together. TrustDroid protects the Android system and attempts to meet confidentiality requirement by isolating the applications and data that are at different trust levels.

Our protection scheme provides a Mandatory access control at the Middleware layer, mainly at the Activity Manager located in the Application Framework layer. Our scheme aims to make the Android users aware of what is happening in their device. For instance, no recording audio leaks, and no private data leaks from their device without user's knowledge. We say that each application can collect private data, or recording audio. It does not penetrate the privacy of users unless this application could leak these information outside the device via Internet (network sockets) or via Bluetooth. However, the Bluetooth differs from the Internet since no application can pair Bluetooth devices without user interaction unless the application has BLUETOOTH_PRIVILEGED permission. Such permission is not available to third party applications. So that, we focus on preventing any application that does not have INTERNET permission to access Internet. Our scheme makes the applications that have a critical permission to be inaccessible by other applications that does not have this critical permission. Then, the system developers select one or more permissions from the permissions categorized as dangerous protection level permissions, and isolate all applications that have one selected critical permission among the applications that does not have the same. For instance, one application has two selected critical permissions, such as READ_CONTACT permission and INTERNET permission, but another just owns one selected critical permission, such as INTERNET permission. Our Firewall will disallow to pass the communication between them if the call initiated from the app has less "critical permissions" but it is allowed if the call initiated from the app has more "critical permissions".

We do not aim to mitigate the confused deputy attacks as IPC inspection that reduces the privilege of the receiving application to the intersection privilege with the caller applications, but we mitigate the confused deputy attacks by aborting any call or Intent message from the caller app that does not have a selected critical permission to the callee app owning this one. Moreover, we handle the collusion attacks also by the same mechanism preventing the confused deputy attacks. Therefore, the difference between our approach and IPC Inspection is obvious.

We made application zoning, but not zoning depending on trust levels as TrustDroid. However, we made zoning depending on the selected critical permissions. Namely, the applications have the same selected critical permissions are located in the same zone and no other applications without this permission can access them.

5 EXPERIMENTAL RESULTS

In order to test the effectiveness of the proposed solution, we performed eight different test cases. Two cases to show the privilege escalation attacks in Android 4.4 reinforced with SE Linux. Six other cases to show the ability of the proposed solution in preventing the privilege escalation attacks. We also performed a comparison between the performance of the Android system before and after adding our proposed solution. More details about these experiments are presented below.

5.1 Privilege Escalation Attack in Android

We created some Android applications to test the possibility of privilege escalation attacks with Android 4.4 where Android sandbox reinforced with SE Linux.

5.1.1 Case 1 (Implementing Confused Deputy Attacks)

Application A is requesting data from a URL via application B. Although application B has an INTERNET permission and application A does not have privilege to access Internet but application A was able to access Internet and get the requested data. In this case, we say that application A and application B achieved confused deputy attack.

5.1.2 Case 2 (Implementing Collusion Attacks)

Application X has no privilege permission, application Y has privilege permission P2, and application Z owns Internet permission and protected by permission P2. Thus application X failed to call directly application Z becuase application X does not have the protection permission of application Z, i.e., P2 but it can do that indirectly via application Y. Then we say that application X, application Y, and application Z cooperated together to achieve collusion attack.

5.2 Privilege Escalation Attack Prevention (Firewall with One Zone)

We used the same application A, application B, application X, application Y, and application Z of the same Android OS after adding our firewall protecting one zone that consists of the applications having INTERNET permission.

5.2.1 Case 3 (Call between Applications at the Same Zone)

We performed the same experiment as case 1 after adding INTERNET permission to application A. Then, application A is able to get the requested data from the Internet through application B. These are accepted behaviors since both applications A and B have the same permission of accessing the Internet.

5.2.2 Case 4 (Preventing Confused Deputy Attacks)

We repeated the previous experiment but after removing the INTERNET permission from the Manifest file of application A. In this case, our firewall aborted the call between application A and application B since application B is protected by our Firewall and has INTERNET permission. Thus, the proposed solution prevents the confused deputy attack.

5.2.3 Case 5 (Preventing Collusion Attacks)

We performed the same experiment as case 2. Since application Z has Internet permission, it is protected by our Firewall. In this case, application X cannot call directly or indirectly application Z. Therefore, the proposed solution prevents the collusion attack.

5.3 Privilege Escalation Attack Prevention (Firewall with two Zones)

We used the same application A, application B, application X, application Y, and application Z with the same Android OS after adding our firewall protecting two zones. One zone consists of the applications having INTERNET permission and another consisting of the applications having READ_CONTACTS permission.

5.3.1 Case 6 (Call between Applications at the Same Zone)

We performed the same experiment in case 1 after adding INTERNET permission and READ_CONTACTS permission to application A and adding READ_CONTACTS permission to application B. Then, Application A is able to get the requested data from the Internet through

application B. These are accepted behaviors since both applications A and applications B have the same permissions. So, they exist at the same zone.

5.3.2 Case 7 (Preventing Confused Deputy Attacks)

We repeated the previous experiment but after removing READ_CONTACTS permission from the Manifest file of application A. In this case, our firewall aborted the call between application A and application B since application B is protected by our Firewall and has READ_CONTACTS privilege permission. Thus, the proposed solution prevents the confused deputy attack disallowing any application outside the zone to access any application inside the zone protected by firewall. Firewall rejects the calls that are initiated from apps own lower critical privileges to apps own higher critical privileges and allows vise versa.

5.3.3 Case 8 (Preventing Collusion Attacks)

We performed the same experiment in case 2 after adding INTERNET permission to application X and adding INTERNET permission and READ_CONTACTS permission to application Y. application X fails to access directly application Z since application Z is protected by protection permission P2 that does not exist in the Manifest file of application X. application X also fails to access indirectly application Z via application Y, because application Y has privileges more than application X's privileges. So that, no way for application X to access application Z except it has the same selected critical permissions or more beside owing the protection P2 permission of application Z as a privilege permission if it wants directly to access application Z.

5.4 Performance Evaluation

In this experiment, we evaluate the runtime performance overhead of Android OS before and after adding our firewall with one zone and two zones. Performance overhead is evaluated using two applications. First, the Benchmark "softweg" tools Application [17] that is available in real devices and used in evaluating SE Android [1]. This tool is available in the Android Virtual Device (AVD) till nearly June 2014. We also used an application, called Benchmark that constitutes an Android Benchmarking tool [18]. We evaluated the performance of Android 4.4 with one zone firewall, and Android 4.4 with two zones firewall by this application. We decided to use different kinds of the performance evaluation applications to obtain an accurate pointer for the comparison between Android 4.4 and Android 4.4 with our firewall protecting one zone or two zones.

The tool measures several performance related parameters. For example, MIPS stands for Million Instructions Per Second. A processor with a lower MIPS score is actually better since its instructions achieve more work per clock cycle [19]. MFLOPS stands for Million Floating point Operations Per Second and is also a measure of computer performance, but it is useful to fields of scientific calculations where there is a large use of floating-point calculations. FLOPS measures the computing ability of a computer, but MIPS is used to measure the performance of integer operation, including data movement (X to Y) or value testing (If X = Y, then Z) [20]. Consequently, MIPS is more suitable in our case to compare between AOSP with and without Firewall.

We run Benchmark softweg tools 20 times and calculate the average of the obtained results. “Table 1” shows the obtained results for Android OS with and without firewall respectively.

Table 1: Benchmark-softweg tools measurement.

Benchmark Title	AOSP	AOSP With Firewall
Total CPU score	103.447	102.0588
MWIPS DP	5.648728	5.828848
MWIPS SP	6.205147	7.223971
MFLOPS DP	1.37835	1.366214
MFLOPS SP	2.086737	1.339957
Total memory score	56.04646	74.09899
Copy memory	50.92819 Mb/sec	67.33193 Mb/sec
Total file system score	17.82758	26.88482
Creating 1000 empty files	17.51805 sec	13.5556 sec
Deleting 1000 empty files	8.19665 sec	7.1975 sec
Write 1 M into file	5.73535 M/sec	7.54123 M/sec
Read 1 M from file	30.1311 M/sec	46.54653 M/sec

Adding our firewall leads to a slight increase in MIPS. There is no effect on MFLOPS as expected from the definition of MIPS and MFLOPS. However, we observe that there is no negative effect of Firewall on copying memory, creating files, deleting files, writing into file, and reading from file. When we use the second tool called Android Benchmarking tool (Benchmark), we run

Benchmark application around 100 times and calculate the average and extracted ones that there are slightly differences between Android 4.4 with one zone firewall and Android 4.4 with two zones firewall. Thus, we can also neglect any negative effect on the performance overhead in case of two zones firewall as we did in one zone firewall. "Table 2" shows the measurement after running Android Benchmarking tool (Benchmarking).

Table 2: Android Benchmarking Tool Measurement.

Benchmark Title	AOSP with one zone FW	AOSP with two zones FW
RAM	40.3625	39.2375
CPU Integer	135.025	135.4
CPU float-point	93.784	93.734
Database IO	403.375	395.7718

6 CONCLUSION AND FUTURE WORK

This paper proposes a solution to prevent the privilege escalation attack on specific permissions. System developers can select certain permissions making them as critical permissions. Thus, all applications of these critical permissions or of one of them are protected by the proposed firewall. No other applications outside the firewall's protected zones will be able to escalate their privilege to one of these critical permissions. The proposed firewall prevents confused deputy attacks and collusion attacks. The proposed firewall is also capable of protecting multiple critical permissions by creating more than one zone.

We implemented and tested the proposed tool in preventing escalation attack for Internet access and reading contacts. Our firewall achieved successfully its task. Based on the performance evaluation, the additional proposed firewall slightly increases the device MIPS.

We observed that the most Android applications require many permissions although they may not need them and this case is defined as excessive privileges attacks and these applications may do unwanted actions threatening the confidentiality, availability, or integrity of the Android operating system without any objection by our firewall, because these applications are granted and confirmed their permissions by the owner user during the installation time.

For instance, many famous Android applications are granted UNINSTALL_SHORTCUT permission and INSTALL_SHORTCUT permission, such as WhatsApp application that may be installed in most Android devices, without any objection by the Android operating system. This application can uninstall a real shortcut and install its fake shortcut then the users applied their data on this new shortcut and this application can capture the user' data and redirect it again. Therefore, the use's privacy and confidentiality are violated. The applications have these permissions, they can also modify in this user's data before redirecting it. Thus, the integrity is violated. This application can deny redirecting the data to the desired target therefore the availability is affected.

We concluded that our modified Android operating system can prevent the confused deputy attacks and the collusion attacks, but the Android is still not operating system secured due to the excessive privilege attacks.

In future work, we will consider other privilege permissions as critical permissions if need and also evaluate the effectiveness and the performance in addition to considering the all permissions exiting in "Manifest.Permission" [21] to set perfect rules facing the android application that owns a set of privilege permissions enabling it to achieve the excessive privilege attacks and threatening the confidentiality, availability, and integrity of the Android operating system. One of the Critics may ask "what is the new in this future work" where for example, The Kirin security service [7] introduces slightly similar solution for mitigating the excessive permissions attacks. The solutions will differ as follows: in which location of operating system the policies (rules) will be enforced, how the policies (rules) will be enforced, and what are these policies (rules). Consequently, the potential of achieving the target successfully is different in addition to the difference in the performance overhead. We will implement the proposed firewall adding to it the new rules that will face the excessive privilege attacks and this implementation will be on real device following the guides [22]. In this case, this solution will handle the confused deputy attacks, collusion attacks, and excessive privilege attacks together. This is a fragment about our solution facing the excessive privilege attacks. We will allow that the applications with excessive privileges are installed on the device such as WhatsApp application for users can enjoy with this application beside securing their privacy by denying any call from this fishy application depending on one of their excessive permissions. For instance, the whatsApp application demand uninstall and install shortcuts permissions to be installed on the device and the user is enforced to accept that to enjoy with this application then we will allow that but if this application attempts to uninstall or install shortcut on the device, our advanced firewall (our rules) just aborts this call.

7 ACKNOWLEDGMENTS

We thank our supervisor Magdy Elsodani for his valued advices and his contribution in this work.

5 REFERENCES

[1] S. Smalley, and R. Craig, "Security Enhanced SE Android: Bringing Flexible MAC to Android", Proceedings of the 20th Annual Network and Distributed System Security Symposium (San Diego, California, USA), February 24-27, 2013. The Internet Society 2013.

[2] Security Enhancements in Android 4.4. https://source.android.com/devices/tech/security/enhancements44.html.

[3] R. MATHEW, "Study of privilege escalation attack on android and its countermeasure", International Journal of Engineering Science and Technology, Sept. 4- 9, 2012, PP. 4078-4082.

[4] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A. Sadeghi, "Xmandroid: A new android evolution to mitigate privilege escalation attacks", Technical Report, Technical University of Darmstadt at Darmstadt, 2011.

[5] Android Architecture. http://www.tutorialspoint.com/android/android_architecture.

[6] I. Kashefi, and M. Salleh, "A survey on mitigating attacks related to shortcomings of android permission framework", Journal of Theoretical and Applied Information Technology. Vol. 55, No. 2, Sept. 2013, PP. 174-182.

[7] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically Rich Application-Centric Security in Android", Proceedings of the 2009 Annual Computer Security Applications Conference (ACSAC '09), The Pennsylvania State University, University Park, PA (USA), Dec. 07-11, 2009, PP. 340-349.

[8] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: trading privacy for application functionality on smartphones", Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile '11), University of Cambridge (United Kingdom), March 01 - 03, 2011, PP. 49-54.

[9] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification", Proceedings of the 16th ACM conference on Computer and communications security (CCS '09), The Pennsylvania State University, University Park, PA (USA), November 09 - 13, 2009, PP. 235-245.

[10] W. Enck, M. Ongtang, and P. McDaniel, "Mitigating Android Software Misuse Before It Happens", Technical Report, Network and Security Research Center, The Pennsylvania State University, University Park, PA (USA), 2008

[11] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, E. Chin, "Permission re-delegation: Attacks and defenses", USENIX Security Symposium, University of California, Berkeley (USA), 2011

[12] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on Android", Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM '11), Darmstadt University (Germany), October 17, 2011, PP. 51-62.

[13] Firewall (computing). http://en.wikipedia.org/wiki/Firewall_%28computing%29.

[14] W. Stallings, "Cryptography and Network Security: Principles and Practice (5th ed.) ", Prentice Hall Press, Upper Saddle River, NJ (USA). 2010.

[15] Android Open Source Project. http://source.android.com/source/downloading.html.

[16] Android Packages Interfaces (APIs). http://developer.android.com/reference/packages.html.

[17] Benchmark softweg - Tools. https://play.google.com/store/apps/details?id=softweg.hw.performance.

[18] Android Benchmarking tool. http://android.downloadatoz.com/apps/com.superapk.benchmark,343436.html.

[19] MIPS/MFLOPS and CPU Performance. http://www.sgidepot.co.uk/perf.html.

[20] FLOPS. http://en.wikipedia.org/wiki/FLOPS.

[21] Manifest permission. http://developer.android.com/reference/android/Manifest.permission.html.

[22] Building for devices. https://source.android.com/source/building-devices.html.