RGBDroid: A Novel Response-Based Approach to Android Privilege Escalation Attacks

Yeongung Park
adminstor@dankook.ac.kr
Dept. of Computer Sci.
Dankook University

ChoongHyun Lee
chl@csail.mit.edu
CSAIL, Dept. of EECS
Massachusetts Institute of Technology

Chanhee Lee
lchan12@dankook.ac.kr
Dept. of Computer Sci.
Dankook Univerisity

JiHyeog Lim
marnitto@dankook.ac.kr
Dept. of Computer Sci.
Dankook Univerisity

Sangchul Han
schan@kku.ac.kr
Dept. of Computer Eng.
Konkuk University

Minkyu Park
minkyup@kku.ac.kr
Dept. of Computer Eng.
Konkuk University

Seong-Je Cho
sjcho@dankook.ac.kr
Dept. of Software Sci.
Dankook University

Abstract

Recent malware often collects sensitive information from third-party applications with an illegally escalated privilege to the system level (the highest level) on the Android platform. An attack to obtain root-level privilege in an Android environment can pose a serious threat to users because it breaks down the whole security system. RGBDroid (Rooting Good-Bye on Droid) is an extension to the Android smartphone platform that effectively detects and responds to the attacks associated with escalation or abuse of privileges. Considering the Android security model, which dictates that users are not allowed to get root-level privilege and that root-level privilege should be restrictively used, RGBDroid can find out whether an application illegally acquires root-level privilege, and does not permit an illegal root-level process to access protected resources according to *the principle of least privilege*. RGBDroid protects the Android system against malicious applications even when malware obtains root-level privilege by exploiting vulnerabilities of the Android platform.

This paper shows that i) a system can still be safely protected even *after* the system security is breached by privilege escalation attacks, and ii) our proposed *response* technique has comparative advantage over conventional *prevention* techniques in terms of operational overhead which can lead to significant deterioration of overall system performance. RGBDroid has been implemented on an embedded board and verified experimentally.

1 Introduction

Android is a Linux based mobile operating system. It strengthens security through basic security mechanisms in Linux. However, it still has security vulnerabilities. Recently, malicious codes exploiting these vulnerabilities have been rapidly increasing [2, 14]. Some malicious Android codes carry out adverse actions with root privilege, the highest administrator privilege in Android, acquired through privilege escalation attacks [5, 6, 13]. Since resource access restrictions of the Android security model [14] can be lifted by such malicious root privilege attacks, an attacker who obtains root privilege can perform various adverse actions. In fact, malwares such as GingerMaster [5], DroidKungFu [6], and DroidDream [13] make the system act as a bot, a compromised computer, by installing a malicious app without notice through the abuse of root privilege acquired by privilege escalation.

This paper presents RGBDroid, a novel security model that provides a fundamentally new approach to protect the system against privilege escalation attacks in the Android environment. Instead of trying to *prevent* privilege escalation attacks, RGBDroid *responds* to the attacks to achieve the same goal (i.e., protecting the system from the adverse actions attainable through privilege escalation).

The Android security model does not allow a user to acquire root privilege directly; it allows only parts of Android systems to acquire limited root privilege. RGBDroid is a security scheme designed to protect a system by detecting and responding to the root privilege acquisition by an attacker or unauthorized apps through privilege escalation attacks. In addition, RGBDroid restricts access with root privilege to resources by strictly applying *the principle of least privilege*. RGBDroid detects unauthorized acquisition of root privilege and restricts access to protected resources by an illegal root-level process using pWhitelist and Criticallist. pWhitelist is a list of normal system programs that can be executed with root privilege. Even if a malicious app acquires root privilege, it cannot execute any programs or access any resources since it is not in the pWhitelist. In addition, RGBDroid uses Criticallist to prohibit a malicious app from affecting user processes' execution directly or indirectly by manipulating resources that are critical to the

operation of the Android system. Criticallist has a list of critical resources that can affect user apps and are vital in operating Android, and protects such critical resources from being manipulated by a root privileged process.

This paper shows that RGBDroid can effectively protect the system against malicious behaviors at the kernel level through an experiment of root shell acquisition and an experiment based on a system resource access scenario. We also show efficiency of the proposed scheme by performance analysis.

This paper is organized as follows. In Section 2, we describe the features of the Android security model and analyze malicious codes that attempt privilege escalation attacks in the Android environment. In addition, we analyze the limitations of the conventional security technologies and explore the needs for a system level security mechanism. Section 3 introduces RGBDroid as a way to mitigate such limitations. In Section 4, we show the effectiveness of RGBDroid through experiments based on actual malicious behavior scenarios. Section 5 analyzes the performance of RGBDroid and we summarize our conclusions in Section 6.

2 Related Work

2.1 The Android Security Model

Android is a Linux-based system. Application programs and system parts execute within their own processes. Android provides functions such as preemptive multitasking, efficient memory management, and user and group control. In addition, its application program privilege mechanism prevents Java applications from abusing the system and resources, and developers are forced to sign their application programs to distribute them [2, 14].

Overall protection between Android applications and the system is based on standard Linux protection mechanism such as assigning a user-ID(UID) or group-ID(GID) to application. Additional protection is implemented through a permission framework that restricts specific operations of processes. The permissions are specified in the AndroidManifest.xml file during application development, and once the development completes the permissions cannot be changed. The user is informed of the specified permissions at installation time. If the user confirms these the app is installed and operates.

As mentioned, Android provides sandboxing based on UIDs and GIDs, file access control of user programs, and account access control using file system configuration. Android also prevents users from using root privilege directly to protect the system against incautious user activities.

2.2 Security Problem in Android

There have been a number of efforts to enhance security in Linux. Examples include TrustedBSD [16], AppArmor [3], TrustedSolaris [4], GRSecurity [8], and SELinux [11]. Most of them aim at precluding illegal resource access and privilege escalation. However every vulnerable point cannot be predicted in various computing environments. In the research by Chen et al. [3], OS level security policies are analyzed and compared, and the authors present attack scenarios that install a rootkit into Ubuntu 8.04 servers under SELinux and AppArmor.

Rooting is exploiting the vulnerability of the Android platform and acquiring root privilege. It includes attacks on the Linux kernel, daemons, and services. Once rooted, normal users can acquire root privilege, make security mechanisms useless, and add or remove arbitrary functions. Enck et al.[2] show that the address book DB can be accessed through an ADB shell after rooting.

Recent privilege escalation attacks on Android exploit trusted programs in a similar way to that in the research by Chen et al. [3]. Privilege escalation attacks may bypass Android security mechanisms and pose a serious threat. In this perspective, a new approach is needed that effectively detects and defeats the attacks related to escalation or abuse of privilege.

2.3 Analysis on malware with rooting

The recent privilege escalation attacks such as DroidKungFu [6], DroidDream [13] and GingerMaster [5] perform malicious actions not only at the user level but also at the system level.

DroidKungFu infects Android market applications. Infected applications add a new receiver component and a new service component. The receiver component is notified when the Android platform completes booting, and then it invokes the service component. Through the service component DroidKungFu collects and sends various pieces of information such as IMEI(International Mobile Equipment Identity), Device ID and SDK version to a remote server, then it tries to obtain a root shell. The root shell receives commands from a C&C(Command and Control) server, and installs a hidden backdoor application. The infected phone is thus converted to a bot.

GingerMaster and DroidDream have some similarities. Both infect normal Android applications. Once an infected application is installed, it registers a service, collects information on the user's device, transfers the information to a remote server and tries to obtain a root shell. The root shell installs another malicious application that receives commands from a C&C server. The malware can successfully evade the detection of most mobile antivirus software.

2.4 Security Solutions in Android

Behavior of most Android malwares using privilege escalation attacks is similar to that of DroidDream, DroidKungFu, and GingerMaster. Most Android security software tries to detect such malware using signature based detection. However, signature based detection is easy to evade, and is useless in detecting unknown malware [12, 15].

Several solutions have been proposed for enhancing Android security. Each solution requires modifying the Android middleware layer and installing one or more extension components. Kirin [18] is an extension component of the Android application installer. It certifies an application based on the application's security policy and Kirin's security policy. Saint [9] adopts a fine-grained access control model and governs install-time permission assignment and run-time use.

TaintDroid [17], based on taint analysis, tracks the flow of privacy-sensitive data. When the data are transmitted over the network, users are notified to identify misbehaving applications. QUIRE [7] is a security solution that can defend against privilege escalation attacks via confused deputy attacks. To address this problem, when there is an Inter Process Communication (IPC) request between Android applications, QUIRE [7] allows the applications to operate with a reduced privilege of its caller by tracking the call chain of IPCs.

Since most security solutions perform in the middleware layer, they can defeat malicious applications in the user layer efficiently. However, it is difficult to detect misbehavior in the system layer. Therefore, a new security solution is needed that can defeat privilege escalation attacks and protect the system layer.

3 RGBDroid Design

This paper presents RGBDroid, a novel security model that adopts a fundamentally new approach called *"postvention"* and provides an effective system protection mechanism by using an Android feature that root privileges are allowed restrictively only to particular programs required to operate Android. RGBDroid is not a *prevention* technique such as those that existing Linux security techniques focus on to prevent privilege escalation attacks, but is instead and effective *response* technique when root privilege is temporarily hijacked by privilege escalation attacks. Temporarily hijacked root privileges signify a condition that an attacker can directly or indirectly manipulate the control flow of a vulnerable program with root privileges. If the control flow of a vulnerable program with root privileges is manipulated by an attacker, activities the attacker wants can be executed by calling particular functions. For example, an attacker

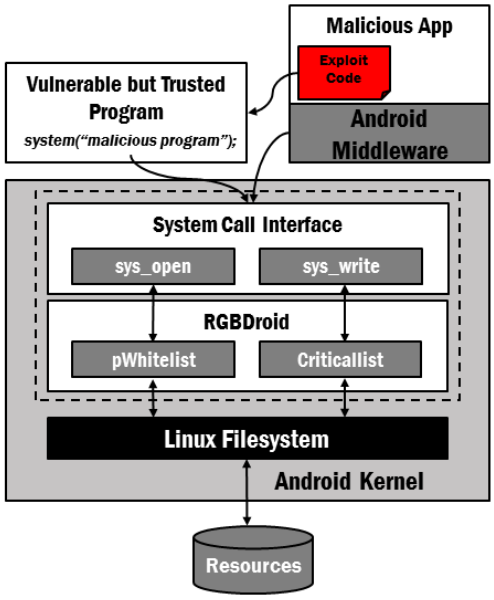

Figure 1: RGBDroid overview

can seize root privileges by manipulating the control flow of a program that has a security vulnerability and then by executing shell code through the system() function. In order to defend these actions, RGBDroid can restrict unlimited access to resources, and detect and respond to the privilege escalation attacks by applying the *principle of least privilege* based on pWhitelist and Criticallist. The overall structure of RGBDroid is shown in Figure 1.

The current Android security system requires the use of a particular interface when a user layer (UID is greater than or equal to 10000) process accesses system layer (UID is smaller than 10000) resources. When a system layer process accesses user layer resources, the Android system controls the access based on the group that the resource owner belongs to according to the Discretionary Access Control (DAC) policy. However, because root privileges can modify and control all parts of the operating system, the current Android security system that does not particularly restrict root privileges can be easily incapacitated by privilege escalation. An attacker can access all system resources unlimitedly by exploiting such a vulnerability. To deal with these problems, RGBDroid is located in the kernel area as shown in Figure 1, and supervises and controls servicing resource access requests from a process. RGBDroid hooks several system calls including open() and write() so that the calls are monitored whenever a suspicious process requests access to core resources. RGBDroid allows or denies access based on pWhitelist and Criticallist. In order to control access to resources, RGBDroid classifies resources in Android into two layers as shown in Figure 2. System layer resources are the resources owned by the accounts whose

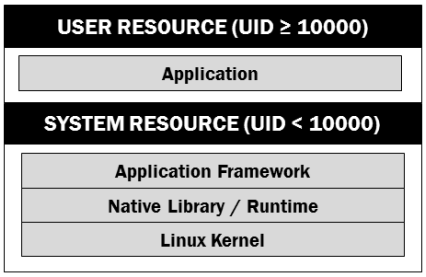

Figure 2: Hierarchical classification of resources

UID(User IDentifier) is less than 10000, and user layer resources are the resources owned by the accounts whose UID is greater than or equal to 10000. Examples of system layer resources are core.jar and framework.jar that are files of library functions used for the Android framework. Examples of user layer resources are the contact list database, SMS, photographs, videos, etc.

RGBDroid systematically monitors and controls access to resources of the two layers depending on the *principle of least privilege*. RGBDroid reinforces Android security features by monitoring and restricting each resource access request that a root privileged app issues. RGBDroid is located between the system call entry and original kernel subsystems, and has been implemented as a Loadable Kernel Module(LKM) to monitor a resource access request of root privilege. By using LKM, the Android kernel does not have to have all possible functionality already compiled into the base kernel; that is, dynamically loading our module into and unloading it from kernel are possible. Inconvenience during development caused by a limitation of root privilege usage and resource access can thereby be prevented. In addition, because modification of the Android platform itself is not required, it is easy to apply RGBDroid to an existing Android system.

Because RGBDroid is implemented through LKM, RGBDroid can be incapacitated or evaded by some types of attacks. A representative example is to unload the LKM module that is already loaded into the kernel using the rmmod system call. RGBDroid detects and blocks such attempts to unload it by hooking the rmmod call. In addition, an attempt to inject malicious code into a process trusted by the pWhitelist is a possible attack, but root privilege is required to attack processes protected by pWhitelist using the ptrace call. It is not easy to perform a particular behavior through code injection via calling the ptrace using temporarily acquired root privilege.

If LKM is disabled in the Android kernel, our proposed solution requires a rooted device and a kernel recompilation. A preferred method is for an Android platform developer to adopt our solution before the final release of their product.

3.1 pWhitelist

pWhitelist is the list of programs that can use root privileges, and RGBDroid permits access to resources requested only by a program with root-level privileges in the pWhitelist. RGBDroid denies any resource access request made by a program which is not a member of pWhitelist. By using such denial of resource request, RGBDroid protects the system against unlimited access to resources by malware with root privileges obtained through a privilege escalation attack.

Because Linux manages each of resources as a file, RGBDroid monitors each file access request made by a program with root privilege by hooking open() system call in kernel level and controls resource access of root privilege using the algorithm shown in Figure 3. As a side note, it is difficult for Linux on PC platforms to adopt an access control scheme using pWhitelist as in RGBDroid; because a root user can directly use root privilege in a PC Linux system, it is not possible to predict which programs will be installed or which resources will be accessed by a program with root privileges. Nevertheless, such software does exist: LIDS (Linux Intrusion Detection System) [1] employs security functionality for PCs which is very similar to that of RGBDroid. LIDS requires a user to directly set access privileges for each resource in the process of kernel compilation.

pWhitelist includes a total 15 processes, and any process not listed in pWhitelist cannot use root privilege. After temporarily obtaining root privilege through a privilege escalation attack, a malicious app needs to run a particular program with root privilege in order to keep the root privilege continuously. At this stage, RGBDroid can effectively defend the system by blocking that particular program with pWhitelist. As a result, a malicious app fails to keep the root privilege that is temporarily hijacked.

The process information in pWhitelist is gathered by logging and analyzing information of processes that uses root privilege. pWhitelist is stored and maintained as an independent file. The list is generated by reading the pWhitelist file during booting phase and kernel initialization. Thus, it is possible to immediately update the process list whenever Android adopts a new process with root privilege.

3.2 Criticallist

Criticallist is a list of system layer resources that even a process with root-level privileges cannot modify. These include critical resources that are essential to operate the Android platform. An attacker who has illegally seized root privilege accesses and manipulates critical resources in system layer can perform malicious activities such as

```
unsigned short uid;
unsigned short euid;

if uid == 0 OR euid == 0
   if !(procname == procname_in_whiltelist)
      return deny;
call sys_open();
```

Figure 3: A hooked open() system call using pWhitelist

```
unsigned short uid;
unsigned short euid;

if uid == 0 OR euid == 0
   if pathname == resource_in_criticallist
      return deny;
call sys_write();
```

Figure 4: A hooked write() system call using Criticallist

Table 1: Protected resources of Criticallist

Resource Name
All the resources of /System/framework directory
/System/etc/hosts
All the resources of /System/lib directory

installing a Managed Code Rootkit [10]. These activities can modify the Android middleware subsystem, and can thus affect the control flow of all programs that run on the subsystem. An attacker can also perform various malicious behaviors by manipulating the configuration files of the Android framework (For example, redirecting a request for a specific URL to the attacker's server by manipulating the /system/etc/hosts file). RGBDroid disallows malware with root privilege to modify system layer resources that do not need to be modified by root privilege and can result in serious adverse effects to the system if manipulated. Note that access to such resources by non-root privilege is restricted by the default Android security mechanism. The system layer resources that are protected by Criticallist in the current version of RGBDroid are shown in Table 1.

The /system directory that contains system resources is mounted as read-only and all resources are owned by root. Therefore, if a non-root user accesses system resources, the access is restricted and controlled by the default access control policy of Android kernel because RGBDroid is built on top of the existing Android security model. However, because an attacker with root privilege can change the read-only permission of the /system directory into readable/writable permissions through remount using mount command, and critical resources under the /system directory can then be manipulated by write operation. If critical resources like DNS are modified, this can cause severe damage to the user or to apps. Criticallist is structured for protecting such important system resources that must maintain integrity. It blocks illegal access to critical resources by wrapping the original write() system call with the algorithm shown in Figure 4.

Using the resource access policy based on Criticallist, RGBDroid can efficiently detect and block unauthorized access to critical resources. As a result, RGBDroid can effectively protects the entire Android system against various privilege escalation attacks by detecting and tightly restricting i) illegal access to critical system layer resources and ii) any further malignant behavior of an attacker who temporarily seizes root privileges.

4 Implementation and Experiments

The experiments were conducted assuming that malicious Android code has temporarily seized root privilege via a privilege escalation attack. Through analysis of existing malicious Android code, we consider both actual and possible malignant activities by malicious apps with root privileges. Then, we prove the effectiveness of RGBDroid by showing those malignant activities fail with our proposed mechanism. Experiments were conducted on an H-AndroSV210 board using Android version 2.2.

4.1 Shell acquisition

According to the Android security model, Android users cannot directly use root privileges and a shell program does not need to be executed with root privileges. However, most of the malicious codes using privilege escalation attacks in Android try to execute a shell program with root privileges in order to maintain their temporarily seized root privileges. If the execution of the shell program with root privileges is not permitted, the attacker cannot maintain root privileges continuously and the privilege escalation attacks eventually fail.

Figure 5 shows a screenshot of the execution of the root shell by connecting H-AndroSV210 through Android Debug Bridge (ADB) when our proposed scheme

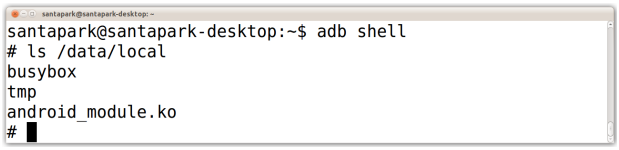

Figure 5: A successful execution of root shell with RGBDroid off

```
santapark@santapark-desktop:~$ adb shell
# /system/bin/sh
link_image[1962]:   940 could not load needed library 'libc.
so' for '/system/bin/sh' (load_library[1104]: Library 'libc.
so' not found)CANNOT LINK EXECUTABLE
#
```

Figure 6: A failure message when attempting to execute shell program with RGBDroid on

is not applied. In Figure 5, we can see the root shell is indeed executed. This demonstrates that malware can obtain root shell access through malicious activity. Figure 6 is a screenshot which shows that the attempt to execute a shell program with root privilege fails in the Android board when RGBDroid is active. Figure 6 shows that access to the libc.so file is denied and thus the execution of the shell program fails. Because the shell program, which is not in the list of processes that can use root privileges, attempts to run using root privilege, pWhitelist in RGBDroid blocks access to the resources required to run the shell program and consequently the attempt fails. In addition, even if a reverse shell is used, the attempt fails for the same reason as above because the shell program must be executed on the target system. This mechanism can block most of the attempts to acquire the root shell in current Android.

4.2 Managed Code Rootkit attack

The managed code rootkit is an attack by manipulating resources required for executing a virtual machine of platform independent languages. Examples of managed code are Java, .NET, PHP, Python, Perl, etc. Android interprets a modified version of the Java language by using a virtual machine called Dalvik which is similar to the JVM. Thus, a managed code rootkit attack may be executed by manipulating base resources for running Android's Dalvik virtual machine. There are a variety of base resources required to run the Dalvik virtual machine, ranging from Java libraries such as Framework.jar and Core.jar to many native libraries located in /system/framework/lib. They are various attack surfaces. The current Android version executes the Dalvik virtual machine without verifying whether such libraries are illegally modified or not, making the rootkit attack possible by manipulating the base resources.

In the experiment, the code for the getByName function in /system/framework/core.jar file is modified to attempt a DNS spoofing attack. Figure 7 shows the result.

The getByName function used for the DNS spoofing attack in Figure 7 is a function that returns the IP address corresponding to a URL received as an argument. In our experiment, we manipulated the code for getByName so that it returns the IP address of www.naver.com if it receives www.victim.com as the URL argument. As shown in Figure 7, the attempt was successful. The managed code rootkit attack shown in Figure 7 inserts a rootkit by replacing the /system/framework/core.jar file with our manipulated core.jar file. By using a root shell we can verify that core.jar file can indeed be replaced (Figure 8). After applying RGBDroid, the attempt to replace core.jar by using the same instructions fails as shown in Figure 9. This occurs because RGBDroid restricts the write operation to the /system/framework/ directory. Manipulation of the various system configuration files in /system/etc directory can similarly be defended.

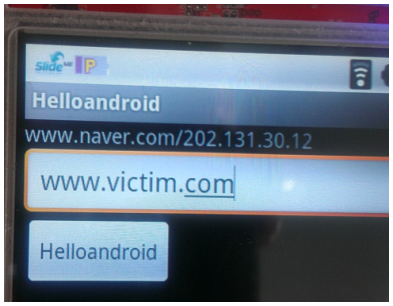

Figure 7: Screenshot that shows successful DNS spoofing attack

```
santapark@santapark-desktop:~$ adb push core.jar /system/fra
mework
3792 KB/s (1862730 bytes in 0.479s)
santapark@santapark-desktop:~$
santapark@santapark-desktop:~$
santapark@santapark-desktop:~$
```

Figure 8: Screenshot showing successful replacement of core.jar file when RGBDroid is off

5 Performance

Performance measurement and evaluation are performed in the same experimental environment as in section 4. We evaluate performance by comparing user program execution time and I/O throughput before and after RGBDroid is applied. Performance measurements were conducted using AndroBench 3.1, an Android storage performance measurement program.

```
santapark@santapark-desktop:~$ adb push core.jar /system/fra
mework
failed to copy 'core.jar' to '/system/framework/core.jar': O
peration not permitted
santapark@santapark-desktop:~$
```

Figure 9: Screenshot showing unsuccessful attempt to replace core.jar file when RGBDroid is on

Table 2: I/O Performance Measurement Table (Unit: TPS (Transactions Per Second))

Count	Before RGBDroid			After RGBDroid		
	Insert	Update	Delete	Insert	Update	Delete
1	25.77	28.17	28.28	24.83	26.56	26.71
2	26.02	28.69	28.1	25.22	26.83	26.67
3	26.14	28.95	28.58	24.84	27.17	23.95
4	26.8	28.72	28.76	23.95	26.36	26.69
5	25.94	28.81	28.3	22.98	26.23	25.36
6	27.4	28.4	28.79	24.78	25.52	26.44
7	24.51	28.67	28.66	23.25	26.69	26.03
8	27.23	27.37	28.5	25.09	27.23	26.89
9	24.49	28.53	27.55	25.03	26.1	26.5
10	26.99	28.73	28.67	25.12	27.33	25.64
Ave.	26.13	28.50	28.42	24.51	26.60	26.09

Table 3: User processing time measurement table (Unit: second)

Count	Before RGBDroid			After RGBDroid		
	Insert	Update	Delete	Insert	Update	Delete
1	11.64	10.64	10.6	12.07	11.29	11.23
2	11.52	10.45	10.67	11.89	11.17	11.24
3	11.47	10.36	10.36	12.07	11.04	12.52
4	11.19	10.44	10.42	12.52	11.37	11.23
5	11.56	10.41	10.59	13.05	11.43	11.82
6	10.94	10.56	10.42	12.1	11.75	11.34
7	12.23	10.46	10.46	12.9	11.23	11.52
8	11.01	10.95	10.52	11.95	11.01	11.16
9	12.24	10.51	10.88	11.98	11.49	11.31
10	11.11	10.44	10.46	11.94	10.97	11.7
Ave.	11.49	10.52	10.54	12.25	11.27	11.51

5.1 I/O throughput

Table 2 shows the results of executing 10 cycles of 300 insert operations, 300 update operations, and 300 delete operations, in SQLite right after Android booted on the H-AndroSV210 board. As shown in the table, the average number of transactions per second (TPS) before applying RGBDroid are 26.13, 28.50, and 28.42 respectively for the three operations and those after applying RGBDroid are 24.51, 26.60, and 26.09. After applying RGBDroid, I/O throughput diminishes by 6.2%, 6.7%, and 8.1% for insertion, update, and deletion respectively. The overall average I/O throughput of the three operations decreases by 7%.

5.2 User Program Processing Time

We measured time consumed for insert, update, and delete operations that the AndroBench 3.1 performed as user program processing time. Table 3 shows that the processing time increases by 6.2%, 6.7%, and 8.4% for each operation after RGBDroid is applied. Average processing time for all three operations increases by 7% overall, which can be considered small processing overhead.

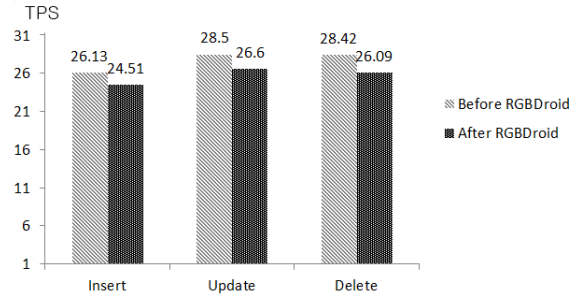

Figure 10: Comparison of I/O performance measurement before and after RGBDroid is activated

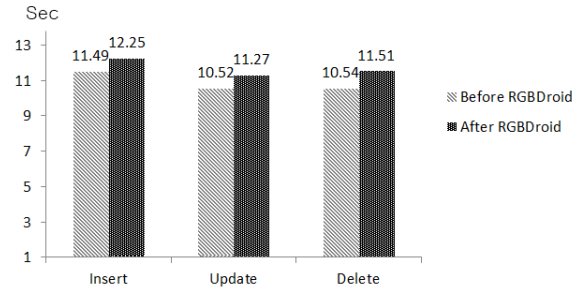

Figure 11: Comparison of user processing time measurement before and after RGBDroid is activated

5.3 Analysis

Table 2 and Table 3 show that the operations required for RGBDroid to control resource access by hooking system call functions related to file I/O do not affect the system performance significantly. Figure 10 and Figure 11 show the graphical comparison of these measurement results before and after RGBDroid is activated. While I/O throughput diminishes by 7% and the user program processing time increases by 7%, the overhead is acceptable considering the level of protection that RGBDroid can provide. Conventional security mechanisms approaches privilege escalation attacks from the perspective of *preventing* them. In order to prevent the attacks, it is necessary to predict potential vulnerabilities in the system. Such prediction requires monitoring and tracing various parts of the system, which inevitably leads to significant performance degradation. More importantly, predicting all possible vulnerabilities is unrealistic in practice as well as in principle. However, the *response based* approach described in this paper, does not rely on the specific knowledge of an individual vulnerabilities. Instead of the high cost of prediction, it responds to attacks by blocking actual adverse actions at the last stage when a root privileged attacker attempts to actually access security-critical or protected resources. In this way, the response based approach emasculates the hijacked root

privileges making them *nominal root privileges*. In summary, after an attacker succeeds in acquiring root privileges temporarily through privilege escalation attacks, the possible activities required to keep root privileges and the potential adverse actions are limited and predictable in contrast with the great many unpredictable potential vulnerabilities in the prevention based approach. Therefore, because the *response based* approach does not require monitoring numerous parts of the system and needs few additional operations, it causes only a small performance overhead unlike the prevention approach.

6 Conclusions

Recent malicious Android codes manipulate system resources or make the system into a bot by seizing root privileges through privilege escalation attacks and stealthily installing malware without notice. This paper presents a solution, RGBDroid, that protects the system by effectively responding after an attacker has already attained root-level privilege. This paper shows experimentally that malware cannot execute specific applications to hold root privileges and that an attacker fails to adversely manipulate system resources. RGBDroid is a fundamentally different approach from the *prevention* approaches that conventional security solutions have used since it protects the system *after* root privileges are hijacked with privilege escalation attacks. Prevention approaches have a high risk of becoming incapacitated because all possible vulnerabilities in the kernel or user applications cannot be predicted. However, the present approach of *responding* to an attack immediately after its occurrence is very efficient because it does not require monitoring or predicting the potential vulnerabilities but just requires blocking possible malicious acts after the attack. Even though limiting root privileges may suggest added inconvenience for users, the security approach of RGBDroid is very suitable and efficient in an Android environment because users do not use root privileges according to Android's security policy. The security measures proposed in this paper present a new approach in security studies, and furthermore, show new possibilities in security research in other operating systems as well as Android.

7 Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0026301) and by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development.

References

[1] ALLEM, T. Lids - deploying enhanced kernel security in linux. Technical report, SANS, Feb 2001. http://www.lids.org.

[2] ENCK W, O. M. M. P. Understanding android security. *Security & Privacy 7* (Feb. 2009), 50–57.

[3] H.CHEN, N.LI, Z. Analyzing and comparing the protection quality of security enhanced operating systems. In *ISOC Conference 2009* (2009), Annual NDSS Symposium'09.

[4] INC., S. M. *Trusted Solaris User's Guide*, 1 ed. Wiley Publishing Inc., 2001.

[5] JIANG, X. Gingermaster: First android malware utilizing a root exploit on android 2.3 (gingerbread). `http://www.cs.ncsu.edu/faculty/jiang/GingerMaster/`.

[6] JIANG, X. Security alert: New sophisticated android malware droidkungfu found in alternative chinese app markets. `http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html`.

[7] M. DIETZ, S. SHEKHAR, Y. P. A. S., AND WALLACH, D. S. Quire: lightweight provenance for smartphone operating systems. In *Proceedings of the USENIX Security Symposium* (2011), Seuciry'11.

[8] M. FOX, J. GIORDANO, L. S., AND THOMAS, A. Selinux and grsecurity: A case study comparing linux security kernel enhancements. `http://cs.virginia.edu/jcg8f/GrsecuritySELinuxCaseStudy.pdf/`.

[9] M. ONGTANG, S. MCLAUGHLIN, W. E., AND MCDANIEL, P. Semantically rich application-centric security in android. In *Proceedings of Annual Computer Security Application Conference* (2009), ACSAC '09.

[10] METULA, E. .net framework rootkits: Backdoors inside your framework. Tech. rep., BlackHat, April 2009.

[11] RUNGE, C. *SELinux: A New Approach to Secure Systems*, 1 ed. Redhat, 2004.

[12] SCHMIDT, A.-D.; BYE, R. S. H.-G. C. J. K. O. Y. K. C. S. A. S. Ststic analysis of executables for collaborative malware detection on android. In *Proceedings of the IEEE International Conference on Communications* (2009), ICC '09, pp. 1–5.

[13] SECURITY, L. M. Lookout mobile security technical tear down - droiddream. Tech. rep., Lookout Mobile Security, 03 2011.

[14] TEAHYUN KIM, HYUNWOOK JIN, I. K. S. P. K. K. S. C. S. H. Analysis of android mobile platform security model. Tech. rep., Korea Internet & Security Agency, University of Seoul, Seoul, Korea, 08 2010.

[15] TROY VENNON, D. S. Threat analysis of the android market. Tech. rep., SMOBILE SYSTEMS, June 2010.

[16] WATSON, R. N. M. Trustedbsd adding trusted operating system feature to freebsd. In *Proceedings of the USENIX Annual Technical Conference* (2001), ATC '01.

[17] W.ENCK, P.GILBERT, B.-G. L. P. C. J. J. P. M., AND SHETH, A. N. Taintdroid: An information-flow tracking system for real-time privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation* (Oct 2010), OSDI'10.

[18] W.ENCK, M. O., AND MCDANIEL, P. On lightweight mobile phone appiication certification. In *Proceedings of ACM CCS* (November 2009), CCS'09.