Detection and prevention of LeNa Malware on Android

Hwan-Taek Lee[1], Minkyu Park[2]*, and Seong-Je Cho[1]
[1] Dankook University, Yongin-si, Gyeonggi-do, Republic of Korea
{htlee, sjcho}@dankook.ac.kr
[2] Konkuk University, Chungju-si, Chungcheongbuk-do, Republic of Korea
minkyup@kku.ac.kr

Abstract

Smartphones contain security-sensitive information of a user such as contacts, *SMS, photos, and GPS information*. Because smartphones are always turned on and ready to connect to the Internet, that sensitive information is in danger of leakage. Various kinds of malware are more and more attacking smartphones, especially Android phones. We propose a scheme that protects Android phones against one of them, called *LeNa*. *LeNa* infects rooted Android phones and periodically leaks sensitive information of the phone. *LeNa* also dominates the system and makes the phone a zombie which can perform Distributed Denial of Service (DDoS) attack. The proposed scheme checks whether a process is allowed to execute a requesting operation even after the process have acquired the root privilege. This scheme can also protect smartphones from malware targeted for rooted phones.

Keywords: LeNa, malware, rooting, Android, root privilege

1 Introduction

On Android, rooting can make a user attain the root privilege, and the user can alter or replace system applications and settings or perform operations unavailable to a normal Android user. In addition, the user can modify the kernel and install custom firmware. On rooted smartphone, user can do a variety of activities whatever he or she wants. Google has been much more relaxed about rooting than Apple has been about *jailbreaking*, thereby many Android users root their phone. Many countries consider Android rooting legal if the purpose is to run legal apps [9].
During rooting, an application named *Superuser* and a program *su* are installed. We can use su to open a root-privileged shell. *Superuser* exchanges information with *su* and can identify the application, which requested the open a root shell. *Superuser* also can ask a user whether she allow or deny the request of *su* [2].
The malicious app *LeNa (LegacyNative)* also uses this su file to open a root shell. *LeNa* gets approving the root privilege from a user as follows: *LeNa* is disguised as an antivirus tool that needs the root privilege, tells the user that if she approves the root privilege, it gives her game items for free, or unlocks a paid application.
LeNa is a kind of social engineering attacks, which attacks rooted smartphones exploiting not vulnerability in Android platform, but psychology of a user. The latest Android phone can be a victim if it is rooted. The network solution company, `Nominum`, reports that *LeNa* is the greatest risk to mobile subscribers on the third in 2013 [10].
We propose a new approach which can detect effectively *LeNa*. The approach utilizes the attack pattern of *LeNa* and characteristics of Android framework and hooks the mount system call and prevents the event that should not occur in a normal usage. Because *LeNa* needs to remount */system* in a *read/write*

*Corresponding author: Room 105, Sanghuh research bldg., Department of Computer Engineering, Konkuk University, 268 Chungwon-daero, Chungju-si, Chungcheongbuk-do, 380-701, Republic of Korea, Tel: +82-(0)43-840-3559

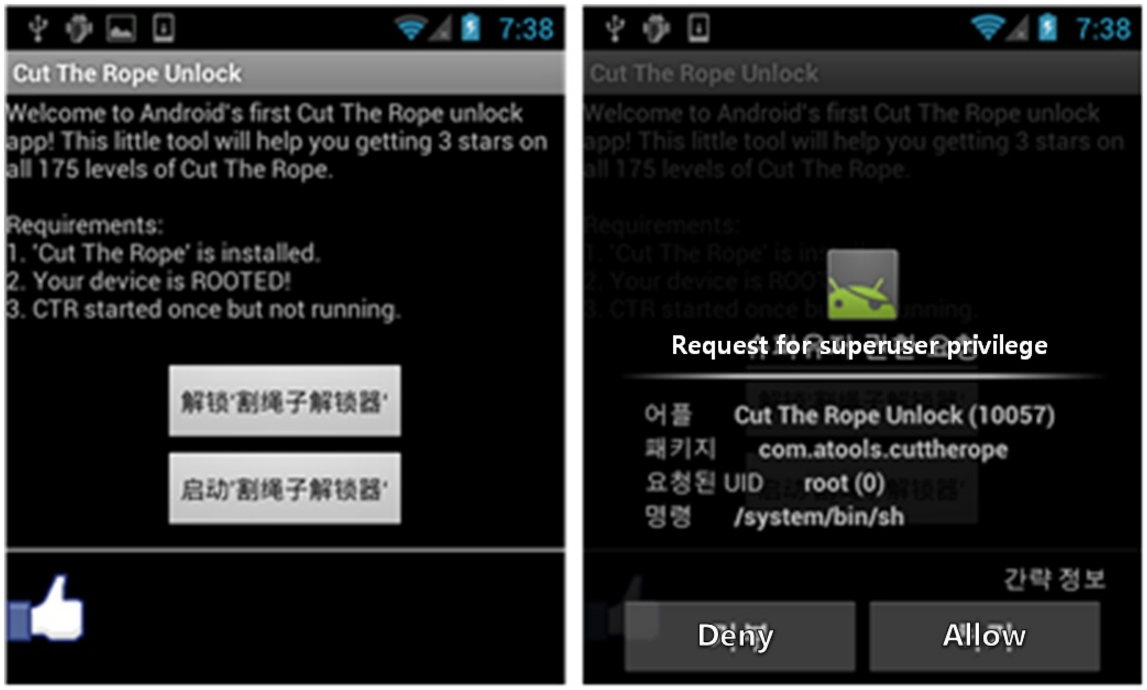
mode, the approach effectively detects attacks by differentiating a normal mount operations and a malicious intended one. We can detect a malicious intended mount operation by checking the mount point and the process that invokes the mount system call. In addition, this approach can also detect malware that target rooted smartphones, such as *GingerMaster* [7] and *DroidKungFu* [8], thus, can be adapted to the lower versions of Android platforms.
This paper is organized as follows. Section 2 describes related work. In Section 3, we analyze a malicious code called LeNa. Then,we explain the environmental features of Android that used in this paper in Section 4, and propose a new method to prevent modification attacks on Android in Section 5. In Section 6, we show the proposed method can also detect another malware. In Section 7, we carry out some experiments to verify the effectiveness of our method. In Section 8, we conclude and give possible future work.

2 Related work

Android's privilege escalation attack can occur in application level and kernel level. *Permission delegation attack* [5], another form of privilege escalation attack, allows an application access the function that it has not permission to use. An application, for example, has not permission to use, GPS service, but the delegation attack make it use by binding it to the service of an authorized application. To prevent such attacks, many studies have been conducted, including *Kirin* [4], *Saint* [11], *QUIRE* [3], and *Xmandroid* [1].
Privilege escalation attack in the kernel level gets the root privileges by exploiting vulnerability of the kernel and takes control of a smartphone. The attack code, then, issues a command to the smartphone through Command and Control server and makes the smartphone a bot. This type of attack is especially fatal to smartphones, because they maintain security sensitive information such as *Address book, SMS, and photographs*.
Park et al. [13] proposed a scheme *RGBDroid*, which prevents malicious behavior after rooting using two additional modules. The first module, *pWhiteList*, lists up all processes allowed to have the root privilege and a process not on the list can never have the root privilege. The second module, *CriticalList*, lists up resources that must not be altered in /system folder and prohibits the removal and change of the resource on the list. The content of *pWhiteList* and *CriticalList*, however, must be modified according to Android version and manufacturer added modules. The proposed approach detects and prohibits a malicious act and is independent of Android version and manufacturers.
Park et al. [12] proposed another scheme, which added *Private Data Protection (PDP)* module to *RGBDroid* above. *PDP* hooks the open system call and checks UID of a process and UID of a file. If the process with the root privilege tries to access resources of the user, *PDP* blocks the open system call. This module prevents all processes with the root privilege from accessing user resources. The proposed approach prevents access with the root privilege, because it targets smartphones rooted by a user herself.

3 LeNa

The *LeNa (LegacyNative)* is a variant of *DroidKungFu* that targets rooted smartphones. While *DroidKungFu* acquires root privileges by exploiting vulnerabilities in Android, *LeNa* does not attack vulnerabilities and is trying to directly attack smartphone rooted by a user. The LeNa installs the management application *Superuser* and adds the root privileged program *su /system/bin* into */system/bin* folder, *Superuser and su* are cooperative. When a process executes *su, Superuser* asks the user whether to give the privilege to the process (Fig. 1).
The early *LeNa* tricks users into approving privilege escalation by displaying the message, which says



Cut The Rope Unlock

Welcome to Android's first Cut The Rope unlock app! This little tool will help you getting 3 stars on all 175 levels of Cut The Rope.

Requirements:
1. 'Cut The Rope' is installed.
2. Your device is ROOTED!
3. CTR started once but not running.

解锁'割绳子解锁器'

启动'割绳子解锁器'

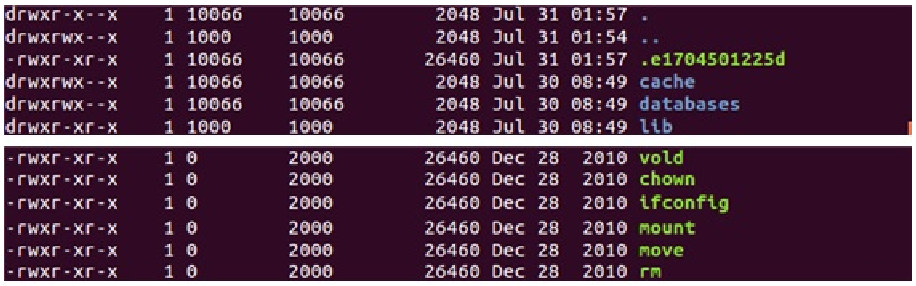
Request for superuser privilege

어플	Cut The Rope Unlock (10057)
패키지	com.atools.cuttherope
요청된 UID	root (0)
명령	/system/bin/sh

간략 정보

Figure 1: The message tricking a user into approving privilege escalation

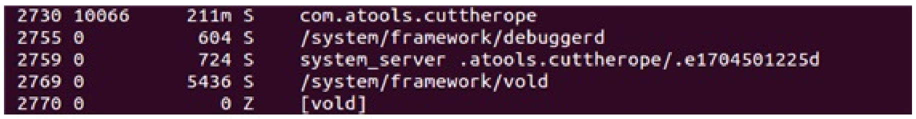
"To use all the features of this application. You have to approve the request for permission". As inducing techniques continue to develop further,*LeNa* disguises itself as an antivirus tool that needs the root privilege, tells the user that if she approves the root privilege, it gives her game items for free, or unlocks a paid application. These tricks are similar to the one used by recent crack programs, which says "The file must be excluded from virus scanning because antivirus tools consider it harmful".
LeNa can open the root privileged shell after your approval. After opening the shell, *LeNa* replaces important files in */system* folder with malicious codes. The top of Fig. 2 shows the *LeNa* application folder and you can see the file *.e1704501225d*. The bottom of Fig. 2 shows the */system/bin* folder. You can find the sizes of important files are same as *.e1704501225d*, including *vold, chown, ifconfig, mount, and rm*. The file move is newly added. The filename *.e1704501225d* is arbitrarily chosen on creation and is deleted after its duty.
In Fig. 3, *vold, debuggerd, and the system_server* have higher PID than *LeNa (com.atools.cuttherope)*. These processes start execution during the booting process and are usually assigned lower PID. After getting the root shell, *LeNa* kills the *vold and debuggerd daemon* processes. The *init* process is the par-



```
drwxr-x--x    1 10066    10066        2048 Jul 31 01:57 .
drwxrwx--x    1 1000     1000         2048 Jul 31 01:54 ..
-rwxr-xr-x    1 10066    10066       26460 Jul 31 01:57 .e1704501225d
drwxrwx--x    1 10066    10066        2048 Jul 30 08:49 cache
drwxrwx--x    1 10066    10066        2048 Jul 30 08:49 databases
drwxr-xr-x    1 1000     1000         2048 Jul 30 08:49 lib
```

```
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 vold
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 chown
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 ifconfig
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 mount
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 move
-rwxr-xr-x    1 0        2000        26460 Dec 28  2010 rm
```

Figure 2: *LeNa* application folder (top) and */system/bin folder* (bottom)



```
2730 10066    211m S    com.atools.cuttherope
2755 0         604 S    /system/framework/debuggerd
2759 0         724 S    system_server .atools.cuttherope/.e1704501225d
2769 0        5436 S    /system/framework/vold
2770 0           0 Z    [vold]
```

Figure 3: Restarted *debuggerd, vold, and system_server*

ent of both daemons. The *init* process receives the *SIGCHLD* signal when they are killed. The init restart two daemons when receiving *SIGCHLD*. Because *LeNa* replaces these files with its malicious codes, the malicious code is executed when they are restarted. The *system_server* process is a child of *zygote* process and usually assigned the PID of *1000 (system)*. *LeNa* changes the parent of *system_server* to *init* using *.e1704501225d* and thus UID of *system_server* is changed into *0 (root)*.

4 The environment of Android

4.1 Remounting /system folder

/system folder contains the critical files to running Android and must not be allowed to change indiscriminately. Android mounts */system* folder in *read-only* mode at the boot time and protects files against deletion and change. If you want to change a file in */system* folder, you need to remount in *read/write* mode. Only a process with the root privilege can remount */system* folder. Android grant the root privilege to the kernel and a few core applications [6]. User application cannot change the content of */system* folder.

4.2 Zygote

Android supports the Java virtual machine on its own, *Dalvik Virtual Machine (DVM)*. The Android applications will run on this *DVM*. It incurs large overhead to initialize the resources needed to run the *DVM* whenever applications are executed, especially on resource-limited smartphones. Android create *zygot*e process to efficiently manage resources.
Zygote is a process that has resources for the *DVM* in memory in advance. When an application is executed, *zygote* forks a child process and adds only information about the applications into the child. This approach does not require significant overhead. Therefore, the parent process of all applications is *zygote* process. The *system_server* also has *zygote* as a parent, thus, does not have the root privilege. Process Hierarchy in Android is shown in Fig. 4.

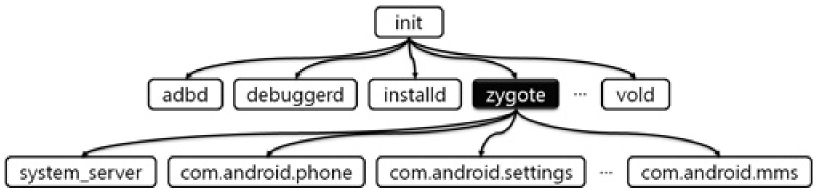


Figure 4: Process hierarchy in Android

5 The Proposed Approach

After acquiring the root privilege, *LeNa* remounts the */system* folder and replaces or creates some important files with its own ones. It's legal for the process with the root privilege to mount the */system* folder,

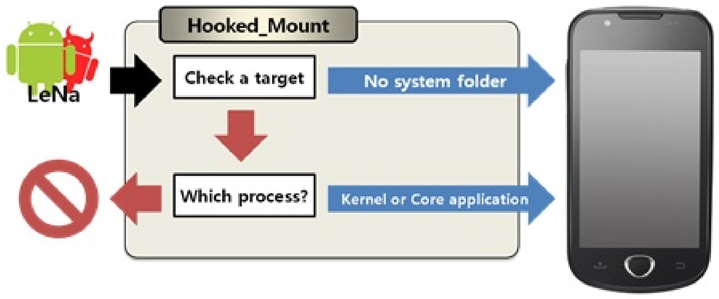
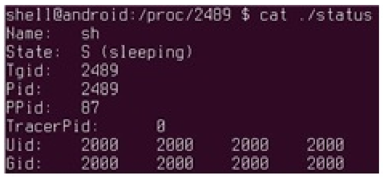


Figure 5: The operation of the hooked mount() system call

but is illegal for a root shell executed by a user application to modify */system* folder. To prevent the user application from remounting */system* folder, we hook the mount system call using Linux loadable kernel module (Fig. 5).

To prevent a root shell executed by a user application from mounting, we must know which process requests the operation and whether the process is a user application or not. We trace the parent process of the requesting process.
Android is on top of the Linux kernel, so */proc* filesystem exits. In the */proc* filesystem, all running processes has the directories named its PID. The directory contains status file, which stores the current status of the corresponding process. The status includes *process's name, UID, GID, PID, PPID, etc*. For example, the content of the status file PID 2489 is shown in Fig. 6.



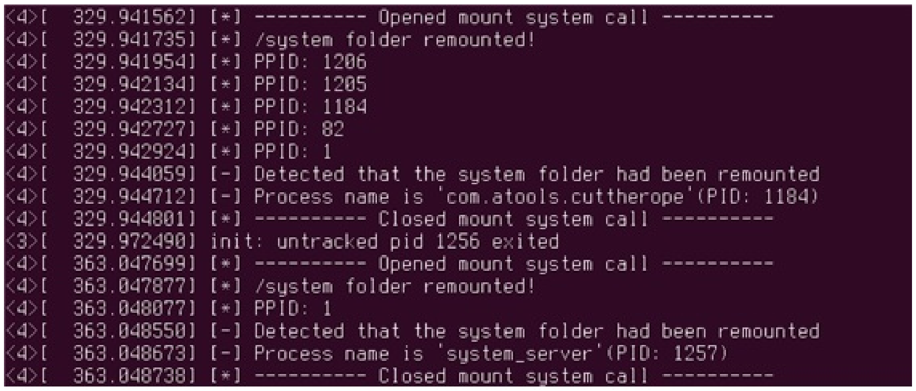
```
shell@android:/proc/2489 $ cat ./status
Name:   sh
State:  S (sleeping)
Tgid:   2489
Pid:    2489
PPid:   87
TracerPid:      0
Uid:    2000    2000    2000    2000
Gid:    2000    2000    2000    2000
```

Figure 6: Process status with PID 2489

If we continue to trace the parent process, we finally reach the init process. Because the legal processes with the root privilege and core system applications are not user applications, *zygote* cannot be their parent process. Only user applications have the *zygote* process as a parent (see section 4.2). If we meet the *zygote* process in the tracing the parent, we can know the requesting process is a user application. Malware *LeNa.a* can be detected by just finding the *zygote* process during the trace. Malware *LeNa.b* makes the init process the parent of the *system_server*, but the *zygote* process should be its parent. If the *system_server* is the child of the *init* process, the fact indicates the *system_server* is a malicious process. *The algorithm of the hooked mount() system call* is shown in Fig. 7.
We verified the effectiveness of the proposed approach by running *LeNa* application on *Android 4.0.4 (Ice Cream Sandwich)* and *Nexus S* smartphone. In Fig. 8, the process *com.atools.cuttherope is LeNa*. *LeNa*'s parent process has PID 82 and it is the *zygote* process. Remounting */system* folder is blocked and no malicious act occurs. Although *LeNa* is running, the legal root privileged processes except the *system_server* is not running (Fig. 9).

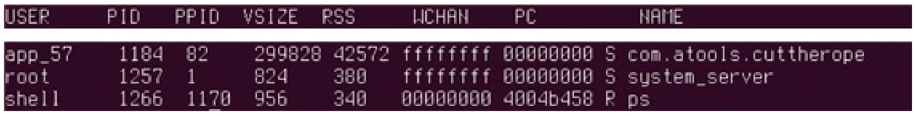
```
hooked_mount(*source, *target, *filesystemtype, mountflags, *data) {
   PID, PPID, processName;
   if(target is /system) {
      PID = getpid();
      while(true) {
         PPID = getppid(PID);
         if(PPID is 1) {
            getProcName(PID, processName);
            if(processName is zygote or system_server) {
               Found malware that remount to system folder!!
               return -1;
            }
            break;
         }
         PID = PPID;
      }
   }
   return orig_mount
}
```

Figure 7: The algorithm of the *hooked mount() system call*



```
<4>[  329.941562] [*] ---------- Opened mount system call ----------
<4>[  329.941735] [*] /system folder remounted!
<4>[  329.941954] [*] PPID: 1206
<4>[  329.942134] [*] PPID: 1205
<4>[  329.942312] [*] PPID: 1184
<4>[  329.942727] [*] PPID: 82
<4>[  329.942924] [*] PPID: 1
<4>[  329.944059] [-] Detected that the system folder had been remounted
<4>[  329.944712] [-] Process name is 'com.atools.cuttherope'(PID: 1184)
<4>[  329.944801] [*] ---------- Closed mount system call ----------
<3>[  329.972490] init: untracked pid 1256 exited
<4>[  363.047699] [*] ---------- Opened mount system call ----------
<4>[  363.047877] [*] /system folder remounted!
<4>[  363.048077] [*] PPID: 1
<4>[  363.048550] [-] Detected that the system folder had been remounted
<4>[  363.048673] [-] Process name is 'system_server'(PID: 1257)
<4>[  363.048738] [*] ---------- Closed mount system call ----------
```

Figure 8: Detection of the illegal mount operation



```
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME

app_57    1184  82    299828 42572 ffffffff 00000000 S com.atools.cuttherope
root      1257  1     824    380   ffffffff 00000000 S system_server
shell     1266  1170  956    340   00000000 4004b458 R ps
```

Figure 9: No malicious act occurs by blocking remounting */system* folder

6 Detecting malware targeting rooted smartphones

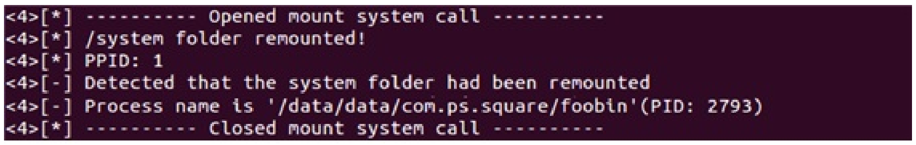
The proposed approach can be used to detect other malware that need to remount */system* folder. It is possible to detect them if we add one more check. a user application is not the direct child of the init

Table 1: Detection results of malware targeting rooted smartphones

Malware	Kaspersky diagnosis	Original	Augmented
DroidDream	Exploit.Linux.Lotoor.l	O	O
DroidKungFu	Backdoor.AndroidOS.KungFu.a	O	O
	Backdoor.AndroidOS.KungFu.bw	O	O
	Backdoor.AndroidOS.KungFu.eh	X	O
	Backdoor.AndroidOS.KungFu.ey	X	O
	Backdoor.AndroidOS.KungFu.hb	O	O
	Backdoor.AndroidOS.KungFu.z	O	O
	HEUR:Backdoor.AndroidOS.KungFu.a	X	O
GingerMaster	Exploit.Linux.Lotoor.z	O	O

process (Fig. 4). The *zygote* process is the direct child of the *init* process and user applications are direct children of the *zygote* process. However, exploiting the vulnerability of *udev* of the *init* process, we can directly execute a root privileged process from the *init* process. *DroidKungFu* use this method to run its own malicious programs. To prevent this bypass, we must check whether the remounting process is in */data/data* folder.

DroidDream and DroidKungFu work only on *version 2.2 or earlier versions* of Android and *GingerMaster* on *2.3.4 or earlier versions*. We test the correctness for this case on *H-AndroSV210 (HyBus_SV210) and Froyo 2.2*. The Table 1 shows the detection result of the original algorithm and the augmented one described above and Fig. 10 shows the detection screen of the augmented algorithm.



```
<4>[*] --------- Opened mount system call ---------
<4>[*] /system folder remounted!
<4>[*] PPID: 1
<4>[-] Detected that the system folder had been remounted
<4>[-] Process name is '/data/data/com.ps.square/foobin'(PID: 2793)
<4>[*] --------- Closed mount system call ---------
```

Figure 10: Detecting illegal remounting of DroidKungFu

7 Performance evaluation

We measured the execution time of the *mount() system call* before and after inserting the kernel module. The module implements the original algorithm. We repeat the measurement *100, 1000, 10000, 100000 times* and calculate the average. The measured execution time of the *mount() system call* is shown in Table 2. The modified *mount() system call* takes twice the time than the unmodified one.

Unlike the mount common for processing either obtain *the PID*, to obtain *the process name* of the PID is added, working time has increased mount when increasing the module. Looking at Table 2, it can be confirmed that the overhead of about two times.

Table 2: The execution time of the mount() system call (s)

The number of repetition	Before	After	overhead
100	0.062366	0.101363	0.038997
1,000	0.363235	0.809746	0.446511
10,000	3.377961	7.909293	4.531332
100,000	33.498014	78.804347	45.306333

8 Conclusion and Future Work

The proposed approach uses the information about the process that tries to remount the */system* folder to determine whether the mount is illegal or not. If a process tries to remount */system*, we trace the parent of the requesting process. Although a process has the root privilege, a user application process cannot mount */system* folder. Therefore, even if a user approved the grant root privileges to *LeNa* unconsciously, no malicious act occurs. In addition this approach can also detect malware that target rooted smartphones, such as *GingerMaster and DroidKungFu*, thus, can be adapted to the lower versions of Android platforms.
The proposed approach blocks all mount of */system* folder of user applications. However, we must consider the case where the user changes the */system* folder directly. We can add the functionality of asking the user whether she wants to allow the mount operation. By this asking, we can selectively block the mount operation.

Acknowledgments

This research project was supported by Ministry of Culture, Sports and Tourism(MCST) and from Korea Copyright Commission in 2013, and by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)".

References

[1] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, and A.-R. Sadeghi. Xmandroid: A new android evolution to mitigate privilege escalation attacks. Technical Report TR-2011-04, Technische Universität Darmstadt, April 2011.

[2] ChainsDD. FAQ. `http://androidsu.com/superuser/faq/`.

[3] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. QUIRE: Lightweight Provenance for Smart Phone Operating Systems. In *Proc. of the 20th USENIX Security Symposium, San Francisco, California, USA*, pages 347–362, August 2011.

[4] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proc. of the 16th ACM conference on Computer and Communications Security (ACM CCS'09), Chicago, Illinois, USA*, pages 235–245. ACM, November 2009.

[5] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission Re-Delegation: Attacks and Defenses. In *Proc. of the 20th USENIX Security Symposium, San Francisco, California, USA*, pages 331–346, August 2011.

[6] Google Inc. Android Security Overview: Rooting of Devices. `http://source.android.com/devices/tech/security/index.html`.

[7] X. Jiang. GingerMaster: First Android Malware Utilizing a Root Exploit on Android 2.3 (Gingerbread). `http://www.csc.ncsu.edu/faculty/jiang/GingerMaster`.

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

[9] N. Kralevich. It's not "rooting", it's openness. `http://android-developers.blogspot.kr/2010/12/its-not-rooting-its-openness.html`.

[10] C. S. Nominum. Top mobile malware threats. `http://nominum.com/blog/mobile-threats/`.

[11] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. *Security and Communication Networks*, 5(6):658–673, June 2012.

[12] Y. Park, C. Lee, J. Kim, S.-J. Cho, and J. Choi. An android security extension to protect personal information against illegal accesses and privilege escalation attacks. *Journal of Internet Services and Information Security (JISIS)*, 2(3/4):29–42, November 2012.

[13] Y. Park, C. Lee, C. Lee, J. Lim, S. Han, M. Park, and S.-J. Cho. Rgbdroid: a novel response-based approach to android privilege escalation attacks. In *Proc. of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats (LEET'12), San Jose, California, USA*, April 2012.

Author Biography

Hwan-Taek Lee received the B.S. degree in Computer Engineering from Dankook University, Korea, in 2013. He is currently a M.E. student in the Department of Software Security at the Dankook University. His research interests include computer security, network security and smartphone security.

Minkyu Park received the B.E. and M.E. degree in Computer Engineering from Seoul National University in 1991 and 1993, respectively. He received Ph.D. degree in Computer Engineering from Seoul National University in 2005. He is now an Associate Professor in Konkuk University, Korea. His research interests include operating systems, real-time scheduling, embedded software, computer system security, and HCI.

Seong-je Cho received the B.E., the M.E. and the Ph.D. in Computer Engineering from Seoul National University, Korea, in 1989, 1991 and 1996 respectively. He was a visiting scholar at Department of EECS, University of California, Irvine, USA in 2001, and at Department of Electrical and Computer Engineering, University of Cincinnati, USA in 2009 respectively. He is a Professor in Department of Software Science, Dankook University, Korea, from 1997. His current research interests include computer security, operating systems, software protection, real-time scheduling, and embedded software.