PtmxGuard: An Improved Method for Android Kernel to Prevent Privilege Escalation Attack

Bin Kong[1, 2], Ying LI[3, 4, *], Lu-Ping Ma[3]

[1]*National Secrecy Science and Technology Evaluation Center, Beijing, China*
[2]*School of Economics and Management, Beijing Jiaotong University, Beijing, China*
[3]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[4]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
Email: liying@iie.ac.cn

Abstract—Vulnerabilities in Android kernel give opportunity for attacker to damage the system. Privilege escalation is one of the most dangerous attacks, as it helps attacker to gain root privilege by exploiting kernel vulnerabilities. Mitigation technologies, static detection methods and dynamic defense methods have been suggested to prevent privilege escalation attack, but they still have some disadvantages. In this paper, we propose an improved method named PtmxGuard to enhance Android kernel and defeat privilege escalation attack. We focus on a typical attack pattern that attacker hijacks the control flow of Android kernel to modify process credentials by corrupting critical global function pointers. PtmxGuard enforces Code Pointer Integrity to Android kernel, checks the accuracy and reliability of those pointers when they're triggered by related system calls, and intercepts the system calls when attack activities are detected. Experiment result demonstrates that PtmxGuard can defense privilege escalation attack effectively.

1 Introduction

Android has become the most popular operation system applied to smart mobile device [1]. Android smart device facilitates people's work and life. Meanwhile, it stores a massive amount of personal privacy and sensitive information, which makes it a preferred target for attacker.

However, vulnerabilities in Android system give opportunity for attacker to damage the system security. Vulnerabilities in Android kernel are considered to be the most dangerous bugs, since kernel is a part of the trusted computing base of Android system [2]. Taking advantage of kernel vulnerabilities, attacker acquires the ability to bypass system protection mechanisms and carry out some malicious behavior, such as gain root privilege and steal personal data. Privilege escalation attack, also known as root exploit, is the process of exploiting kernel vulnerabilities to obtain the highest privilege [3]. If a process gains the root privilege, it has access permission to access any data and obtains control over the whole system.

Many efforts have been made to mitigate kernel vulnerabilities and prevent privilege escalation attack. Security-Enhanced Linux (SELinux), Data Execution Prevention (DEP), Privileged eXecute Never (PXN) and Address Space Layout Randomization (ASLR) have been introduced to enhance Android system and raise the difficulty of vulnerability exploit. However, SELinux can be broken through by corrupting the security identifier and security context of process credentials, and real attacks show that mitigation technology can be bypass by other exploit technology, e.g. Return-Oriented Programming (ROP) [4-9]. Some researchers suggested static detection methods to recognize privilege escalation attack [10, 11], but they all rely on the source code and predefined features. There are some other dynamical approaches to defeat privilege escalation attack, such as PREC, RGBDroid and Security Identifier Randomization [12-16]. Although they have contributed significant improvement on protecting Android kernel, they also have some certain limitations, which are described in Section II..

In this paper, we presented an improved method to enhance Android kernel and defense privilege escalation attack. Firstly, we studied on a number of attack instances and find out a typical attack pattern that attacker tampers critical global pointers, hijack the control flow of kernel and finally modify the process credentials. Secondly, we proposed an improved method named PtmxGuard to protect Android kernel from privilege escalation attack. PtmxGuard enforces Code Pointer Integrity to Android kernel. We specifically focus on the critical global function pointers defined by ptmx driver, which are most likely to be utilized by privilege escalation attack. PtmxGuard verifies the accuracy and reliability of those pointers when they are triggered by related system calls, and intercepts root exploit as soon as it detected attack activity. Finally, experiment result shows that PtmxGuard can defense privilege escalation attack effectively.

The paper is organized as follows. Related work is discussed in Section II. In Section III, we explain that the aim of privilege escalation attack is tampering the UID of malicious application to zero. We summarize a typical pattern of privilege escalation attack and introduce how to trigger a function pointer by related system call. Section IV shows the proposed improved method named PtmxGuard and presents the details of how to defense privilege escalation attack on the base of Code Pointer Integrity. Experiment and result of the proposed method is shown in Section V. Conclusion is made in Section VI.

2 Related Work

Many efforts have been made to mitigate Android kernel vulnerabilities and protect the system against privilege escalation attack.

SELinux has been introduced into Android kernel and has been enforced since Android version 4.4 [4]. Instead of Discretionary Access Control (DAC), SELinux adopts flexible Mandatory Access Control (MAC) mechanisms to provide fine-grained authorization according to pre-configured security policy. However, an attacker can tamper the process's security identifier and security context, and break through the restriction of SELinux's policy. DEP technique enables the kernel to mark certain areas of memory as non-executable, such as Dalvik heap, which can limit executing injected malicious code [5]. PXN has been introduced into Android system to prevent the kernel return to user space to execute untrusted instructions [6]. Unfortunately, real attacks show that DEP and PXN can be fully bypassed by code reuse attack, such as ROP, which leverages existing native code in memory as shellcode payload instead of injecting custom malicious instructions [7, 8]. ASLR is supported since Android kernel version 2.6.35 [9]. It allows the kernel to randomize the stack address, heap address, the location of shared libraries and other key memory areas. ASLR makes it probabilistically hard for attacker to locate key variables. The vulnerability mitigations above enhance the system's ability to resist privilege escalation attack. Adding mitigation techniques makes kernel vulnerabilities more difficult to exploit than they would be without mitigations. However, they can't prevent privilege escalation attack completely.

Some methods have been proposed to detect and resist privilege escalation attack. Won-Jun Jang et al. suggested a mechanism to detect rooting attack by monitoring inter process communication, extracting the characteristics of decompiled code and recognizing abnormal activities [10]. However, this mechanism may produce false alarm, as the rooting attack features it identified is too broad. DroidExec is a root exploit malware recognition by feature matching based on similarity calculation [11]. The recognition rate relies on decompile technology and the predefined features. Those two method perform static detection. In contrast, PtmxGuard is able to detect privilege escalation attack dynamically.

PREC is a light-weight framework to dynamically identify system calls originated from third-party native code, and execute them within isolated threads [12]. PREC defeats privilege escalation attack by slowing down the execution frequency of those system calls. It focuses on those malicious activities that repeat high-risk system calls for many times in a short time interval. Yeongung Park et al. proposed a security enhancement framework for Android kernel. It introduces three mechanisms to limit untrusted programs with root privilege to access protected critical system resources and user's private data [13-15]. However, the framework takes effect when an attack program has obtained root privilege illegally and is trying to access some critical and privacy data. In contrast, PtmxGuard doesn't give an attacker a chance to get root privilege. PtmxGuard intercepts the privilege escalation attack, as soon as it detected those malicious behaviors. Lifeng Wei et al. proposed an improved method to enhance Android kernel. By replacing the ways of allocating security identifier from sequential allocation to randomized allocation, it becomes hard for attacker to determine the security identifier of root user [16]. However, the protection may be compromised by exhausting the value of security identifier.

3 Privilege Escalation Attack

3.1 Android User ID

Android system allocates a distinct system identity called User ID (UID) to each application, and Android kernel regards each application as a unique Linux user. Application Sandbox Mechanism isolates each application in its own process space and restricts access to resources by UIDs, prevent applications from interfering with each other at run time.

Generally, the UIDs of user applications are always greater than 10,000. Android kernel retains certain fixed UIDs for system applications, e.g. 1000 for system server process. In particular, a zero UID is associated with the root user, who is unconfined and owns the highest access permissions in Android kernel. Thus, the aim of privilege escalation attack is tampering the UID of malicious app to zero.

3.2 Android Process Credential

In Android kernel space, each process is associated with a structure of process credential named cred. It stores all security attributes of the corresponding application process, including four pairs of UIDs/GIDs: the real (*uid/gid*), the saved (*suid/sgid*), the effective (*euid/egid*) and the filesystem's (*fsuid/fsgid*). If SELinux is configured in kernel, cred will include a *security* pointer points to the structure *task_security_struct*, which contains the SELinux security context (e.g. *sid* and *osid*) of the process.

Android system provides a kernel stack for each process in order to maintain some necessary information when the process switches to kernel space via system call and runs in kernel mode. A process's kernel stack and its process control block *thread_info* store in a union and share the same memory area, which is normally 8KB aligned. The structure *thread_info* has fixed size and stores at the beginning of the shared memory. It contains a *task* pointer points to the process descriptor *task_struct*. The structure *task_struct* includes a *stack* pointer points to the start of the shared memory. There are also two critical pointers *cred* and *real_cred*, and they both point to the structure *cred*. Fig. 1 shows the relationship between several data structures mentioned above.

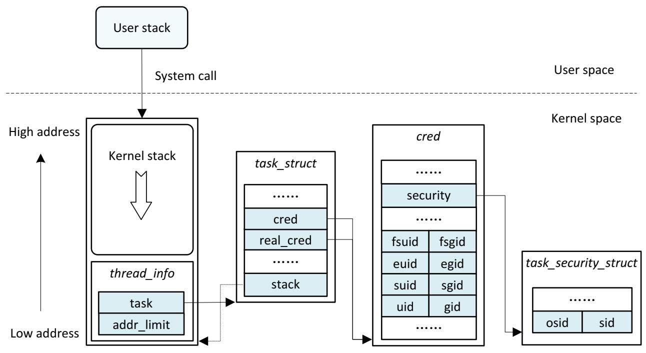


Figure 1. Android process credential

3.3 Typical Pattern of Privilege Escalation Attack

The aim of privilege escalation attack is tampering the UID of malicious app to zero, which represents the root user of Android kernel. However, user application is not allowed to modify variables of process credentials saved in kernel space. Therefore, attacker exploits kernel vulnerabilities to achieve privilege escalation.

Attacker usually carefully construct input parameters of vulnerability function to overwrite something critical, e.g. global function pointers in the kernel, manipulate them to his liking and hijack the control flow of kernel by triggering pointer.

After studied a number of root exploits, we include a typical pattern of privilege escalation attack, as shown in Fig. 2.

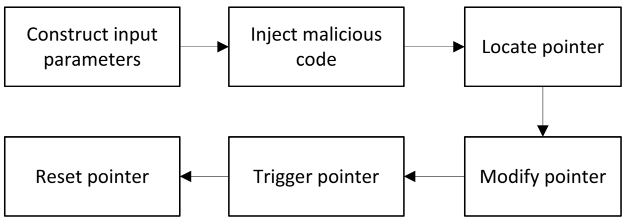


Figure 2. Typical pattern of privilege escalation attack

Firstly, construct input parameters and pass them to the vulnerability function. Crafting input data allows manipulating the target memory region to an expected state. Secondly, allocate an executable memory region and inject malicious code. The injected code is used to modify security attributes (e.g. *uid, sid*) stored in the process credential of the attack application. Thirdly, find an available function pointer in kernel space and get its position. Available function pointers have the ability to manipulate kernel's execution flow. Then, exploit the vulnerability and modify the function pointer to point to the injected malicious code. After that, trigger the function pointer by related system call. The kernel function associated with the function pointer will be performed, and the kernel is hijacked and turns to execute the injected code. Finally, reset the function pointer if necessary to avoid unexpected situation.

The most commonly used pointers is *fsync* of the structure *ptmx_fops*, which is a global structure in the kernel symbol table. Due to the way the binary kernel image is loaded, all global items in the kernel symbol table have the same static address, even if the system reboots.

3.4 Android System Call

The key step to execute the injected malicious code is triggering the modified function pointer by related system call. Android system call is a set of standard interface providing by kernel for restricted communication between user space and kernel space. Applications in user space switch to kernel model to access kernel resource by system call.

As shown in Fig. 3, application in user space invokes system-call interface wrapped by C/C++ library and pass necessary parameters. Then, the system-call interface invokes the intended system call in kernel space. After that, system call invokes kernel function associated with the modified function pointer to access resource, and return states or values to the upper level.

DOI: 10.1051/itmconf/20171205010

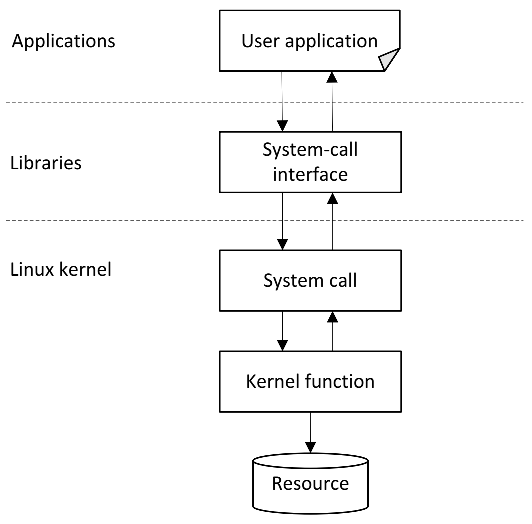


Figure 3. Procedure of system call

4 Design and Implement

According to the typical attack pattern mentioned above, attacker gains privilege escalation by corrupting critical function pointers, and hijacking the control flow of Android kernel to execute injected malicious code. After modifying function pointer, attacker retrieves it and triggers the pointer by related system calls. Therefore, triggering function pointer by related system call is the key to achieve hijack. Hence, we introduce Code Pointer Integrity when triggering the pointer to resist this type of privilege escalation attack.

4.1 Framework Design

An improved method PtmxGuard is introduced into Android kernel to defense privilege escalation attack. PtmxGuard enforces Code Pointer Integrity when triggering critical function pointers, checks whether they are accurate and reliable. If not, PtmxGuard will recognized it as an attack and intercept it.

Fig. 4 shows the framework of the proposed PtmxGuard. It contains five function modules: the monitor module, the sensitivity check module, the integrity verification module, the attack detection module, and the intercept module. There are also two data sets: the sensitive function set and the attack feature set. The sensitive function set includes critical kernel functions, which are most likely to be utilized by privilege escalation attack. And attack feature set defines some typical features extracted from real privilege escalation attack instances.

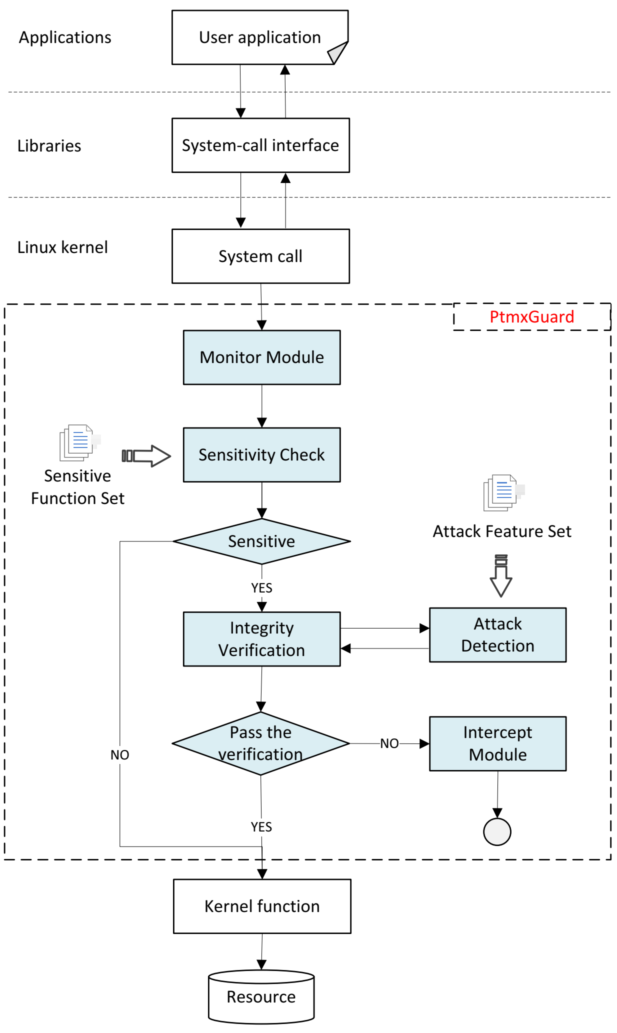


Figure 4. Framework of the proposed PtmxGuard

When an attacker tries to trigger the modified function pointer and invoked corresponding kernel function by related system call, the monitor module will detect the behavior dynamically and the sensitivity check module will retrieve the sensitive function set to check whether the corresponding kernel function is sensitive. If not, PtmxGuard allows the system call to invoke insensitive kernel function. Otherwise, the integrity verification module will retrieve the value of function pointer and pass it to the attack detection module to match the attack feature set. If the memory region where the function pointer points to fits the attack features, it will not pass the verification. PtmxGuard regards it as an attack and the Intercept Module will send a warning and intercept the system call.

4.2 Implement Details

PtmxGuard contains five function modules and the implement details are shown in below.

1) Monitor Module

Using hook technique to monitor system calls and check whether they intend to invoke kernel functions. When detecting the behavior, it delivers to the next module.

2) Sensitivity Check Module

Query the sensitive function set to judge whether the intended kernel functions are sensitive and associated with privilege escalation attack. If not, allow system calls to invoke kernel functions directly. Otherwise, go to the next module.

We define the kernel functions related with the *ptmx* driver as sensitive functions and include them into the sensitive function set. The driver *ptmx* is defined in Android kernel and represented by a kernel-level file structure *ptmx_fops*, which holds pointers to functions defined by the driver. Function pointers within *ptmx_fops*, particularly the pointer *ptmx_fops.fsync*, are preferred targets for attacker, since the values of most entries within *ptmx_fops* are 0 by default, except the pointer *ptmx_fops.open*. Therefore, reset pointer is relatively uncomplicated and only requires writing the value 0 back to the corrupted pointer location.

We define the sensitive function set as shown in TABLE I, which includes sensitive kernel functions and corresponding function pointers.

TABLE I. Sensitive Function Set

Kernel Functions	**Function Pointers**
ptmx_fops->fsync()	ptmx_fops.fsync
ptmx_fops->llseek()	ptmx_fops.lseek
ptmx_fops->read()	ptmx_fops.read
ptmx_fops->write()	ptmx_fops.write
ptmx_fops->ioctl()	ptmx_fops.ioctl

3) Integrity Verification Module

PtmxGuard monitors system calls and determines that one of which is trying to invoke sensitive kernel function. After that, this module verifies the integrity of corresponding function pointer. It firstly gets the function pointer from the sensitive function set, retrieve its value and check whether it is empty. The values of all pointers in Table 1 are set to NULL or 0 by default. If not, we think it is occupied and pass the value of pointer to the attack detection module for further verification and obtain the return result. If the return result implies that the pointer is accurate and reliable, PtmxGuard allows the system call to invoke sensitive kernel function. Otherwise, deliver it to the intercept module.

4) Attack Detection Module

The module is responsible for attack detection. It fetches instructions in the memory region where the function pointer points to, and checks whether they match the features according to the attack feature set.

We extract three attack features from real privilege escalation attack instances as follows:

- The function pointer points to user space. Kernel function pointers are restricted within kernel space in consideration of system security. Pointing to user space is regarded as dangerous, since user space is free for user applications but out of control of Android kernel. The address range of user space is from 0x00000000 to 0xbfffffff.
- The function pointer points to the heap memory region. Heap memory is marked as non-executable by default because of Android Data Execute Prevention. Kernel function pointers will not point to non-executable memory region unless they are tampered.
- The instructions in where the function pointer points to have the ability to modify security attributes of process credentials. The most common instruction is *commit_creds(prepare_kernel_cred(0))*, which allocates a *cred* struct with full privileges as the same as root user and then applies it to the current process.

New attack feature will be added to improve the proposed PtmxGuard, if new type of privilege escalation attack appears.

5) Intercept Module

If system call fails to pass the integrity verification, this module will log related information, intercept the system call and reset the occupied pointer to empty value in order to avoid unexpected situation.

5 Experiment

In order to prove the defense effect of PtmxGuard, we conduct experiment in Android virtual device. We choose LG Nexus 5 for experiment. The detail of experiment environment is shown in Table II.

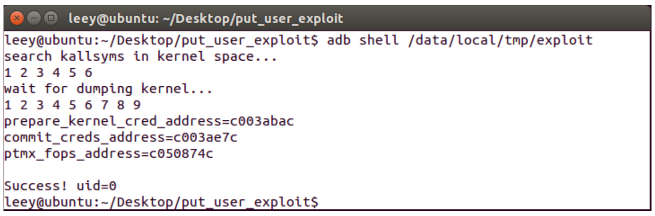
TABLE II. Experiment Environment

Hardware Configuration	Model	LG Nexus 5
	CPU	Goldfish 3.4
	ROM	2GB RAM, 16GB ROM
	Architecture	ARMv7
Software Configuration	Android Version	4.2.2
	Kernel Version	3.4.67

We pick the kernel vulnerability CVE-2013-6282 as example. It’s a typical privilege escalation vulnerability exposed in Linux kernel and also applicable for Android kernel. By exploiting the vulnerability, attack modifies the pointer *ptmx_fops.fsync*, and hijack kernel to execute injected code to gain privilege escalation by invoking corresponding kernel function *ptmx_fops->fsync()*.

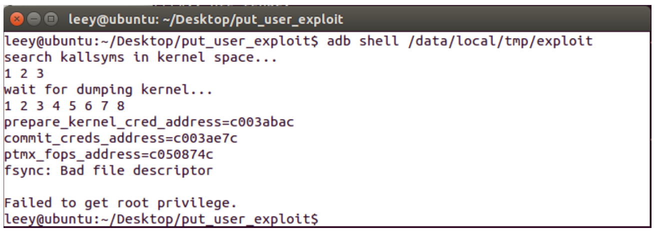
We develop an exploit program named put_user_exploit to prove the defense effect. Before introducing the proposed PtmxGuard, the exploit program succeeds in privilege escalation and sets the *uid* of current process to 0, as shown in Fig. 5. After implementing PtmxGuard, the exploit program fails to get root privilege, because the malicious behavior of put_user_exploit is detected and intercepted by PtmxGuard, as shown in Fig. 6.

Fig. 7 shows the log information recorded by PtmxGuard. In this figure, the value of the sensitive function pointer *ptmx_fops.fsync* is 0x84cc, which locates in user space instead of kernel space. Therefore, it can't pass the integrity verification and intercepted by PtmxGuard.


```
leey@ubuntu:~/Desktop/put_user_exploit$ adb shell /data/local/tmp/exploit
search kallsyms in kernel space...
1 2 3 4 5 6
wait for dumping kernel...
1 2 3 4 5 6 7 8 9
prepare_kernel_cred_address=c003abac
commit_creds_address=c003ae7c
ptmx_fops_address=c050874c

Success! uid=0
leey@ubuntu:~/Desktop/put_user_exploit$
```

Figure 5. Succeed in privilege escalation without PtmxGuard


```
leey@ubuntu:~/Desktop/put_user_exploit$ adb shell /data/local/tmp/exploit
search kallsyms in kernel space...
1 2 3
wait for dumping kernel...
1 2 3 4 5 6 7 8
prepare_kernel_cred_address=c003abac
commit_creds_address=c003ae7c
ptmx_fops_address=c050874c
fsync: Bad file descriptor

Failed to get root privilege.
leey@ubuntu:~/Desktop/put_user_exploit$
```

Figure 6. Fail to get root privilege with PtmxGuard

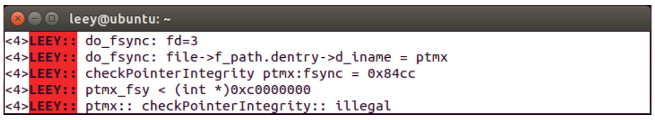

```
<4>LEEY:: do_fsync: fd=3
<4>LEEY:: do_fsync: file->f_path.dentry->d_iname = ptmx
<4>LEEY:: checkPointerIntegrity ptmx:fsync = 0x84cc
<4>LEEY:: ptmx_fsy < (int *)0xc0000000
<4>LEEY:: ptmx:: checkPointerIntegrity:: illegal
```

Figure 7. Log information of PtmxGuard

Conclusion

In this paper, we propose an improved approach named PtmxGuard to enhance Android kernel and defeat privilege escalation attack. We focus on a typical attack pattern that attacker hijacks the control flow of Android kernel by corrupting critical global function pointers, and force the kernel to modify security attributes of process credentials. PtmxGuard enforces Code Pointer Integrity to Android kernel, checks whether they are accurate and reliable when triggered by related system calls, and intercepts those system calls when attack activities are detected. Experiment result shows that PtmxGuard can defense privilege escalation attack effectively.

References

[1] Gartner. Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016[EB/OL]. [2017-01-12]. http://www.gartner.com/newsroom/id/3415117.

[2] Chen H, Mao Y, Wang X, et al. Linux kernel vulnerabilities: State-of-the-art defenses and open problems[C]. Proceedings of the Second Asia-Pacific Workshop on Systems. ACM, 2011: 5.

[3] Vidas T, Zhang C, Christin N. Toward a general collection methodology for Android devices[J]. digital investigation, 2011, 8: S14-S24.

[4] Smalley S, Craig R. Security Enhanced (SE) Android: Bringing Flexible MAC to Android[C]. NDSS. 2013, 310: 20-38.

[5] Drake J J, Lanier Z, Mulliner C, et al. Android hacker's handbook[M]. John Wiley & Sons, 2014.

[6] Xu W, Fu Y. Own your android! yet another universal root[C]. 9th USENIX Workshop on Offensive Technologies (WOOT 15). 2015.

[7] Kanonov U, Wool A. Secure Containers in Android: the Samsung KNOX Case Study[J]. arXiv preprint arXiv:1605.08567, 2016.

[8] Davi L, Dmitrienko A, Sadeghi A R, et al. Privilege escalation attacks on android[C]. International Conference on Information Security. Springer Berlin Heidelberg, 2010: 346-360.

[9] Liebergeld S, Lange M. Android security, pitfalls and lessons learned[M]. Information Sciences and Systems 2013. Springer International Publishing, 2013: 409-417.

[10] Jang W J, Cho S W, Lee H W, et al. Rooting attack detection method on the Android-based smart phone[C]. Computer Science and Network Technology (ICCSNT), 2011 International Conference on. IEEE, 2011, 1: 477-481.

[11] Wei T E, Tyan H R, Jeng A B, et al. DroidExec: Root exploit malware recognition against wide variability via folding redundant function-relation graph[C]. 2015 17th International Conference on Advanced Communication Technology (ICACT). IEEE, 2015: 161-169.

[12] Ho T H, Dean D, Gu X, et al. PREC: practical root exploit containment for android devices[C]. Proceedings of the 4th ACM conference on Data and application security and privacy. ACM, 2014: 187-198.

[13] Park Y, Lee C, Lee C, et al. Rgbdroid: a novel response-based approach to android privilege escalation attacks[C]. Proceedings of the 5th USENIX conference on Large-Scale Exploits and Emergent Threats, LEET. 2012, 12: 9-9.

[14] Park Y, Lee C, Kim J, et al. An Android security extension to protect personal information against illegal accesses and privilege escalation attacks[J]. Journal of

Internet Services and Information Security (JISIS), 2012, 2(3/4): 29-42.

[15] Lee C, Kim J, Cho S, et al. Unified security enhancement framework for the Android operating system[J]. The Journal of Supercomputing, 2014, 67(3): 738-756.

[16] Wei L, Zuo Y, Ding Y, et al. Security Identifier Randomization: A Method to Prevent Kernel Privilege-Escalation Attacks[C]. 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA). IEEE, 2016: 838-842.