Libra: An Adaptive Method for Protecting Memory from Arbitrary Overwrite

Chun-Yi Wang[1]*, Chieh-Wei Huang[1], Fu-Hau Hsu[1], Shih-Jen Chen[2], Yao-Hsin Chen[3]

[1] Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan, R.O.C.
[2] Institute for Information Industry, Taipei, Taiwan, R.O.C.
[3] Information and Communication Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan, R.O.C.

* Corresponding author. Tel.: 886-987898191; email: 101582016@cc.ncu.edu.tw
Manuscript submitted April 24, 2015; accepted September 25, 2015.
doi: 10.17706/jcp.11.4.280-288

Abstract: There have been more vulnerabilities in the Linux kernel in 2013 than there had been in the previous decade. In this paper, the research was focused on defending against arbitrary memory overwrite in privilege escalation. To avoid malicious users getting root authority, the easiest way is to set the sensitive data structure to read-only. But we are not sure the sensitive data structure will never be modified by legal behavior from a normal device driver; thus, we posed an adaptive solution between read-only solutions and writable solutions which is based on the mechanism of read-only IDT table to enhance compatibility. The main idea that we posed not only solves the above problem, but also the general problem which is ensuring that important memory values can only be changed within a safe range. It is not just set to read-only. In addition, we do not need to care about if the Linux kernel exists any the vulnerabilities of arbitrary memory overwrite.

Key words: Arbitrary memory overwrite, exploit, Linux kernel vulnerabilities, privilege escalation.

1. Introduction

The vulnerabilities of Linux kernel have more than the past decade in 2013. In this paper, the research was focus on defending against arbitrary memory overwrite of privilege escalation. Privilege escalation is the action of exploiting a bug in an OS or a software application to gain the root access right. The market of mobile devices has increasingly been raised in recent years; as a result, more attackers turned their attention to mobile devices such as smart phone, tablet, etc.

For example, DroidKungFu [1], GingerMaster [2], and DroidDream [3] perform malicious actions not only at the user level but also at the system level. They make the system act as a bot, a compromised computer, by installing a malicious app without noticing through the abuse of root privilege acquired by privilege escalation. RGBDroid [4] is a response-oriented security solution in 2012. It differs from prevention-oriented security solutions. At first the response-oriented security defines critical malicious behaviors to be potential dangers under the assumption that the Android system was compromised by attacker. The developer of RGBDroid made a responsible policy for each defined malicious behavior considering the features of the Android system. Android is a mobile operating system based on Linux kernel, and an exploit which used the arbitrary memory overwrite of privilege escalation (CVE-2013-2094) was found on the Samsung Galaxy Note

I717 [5].

A classical exploit for arbitrary memory overwrite of privilege escalation is described that the attacker looks for the IDT table or system symbol table to get the pointer which they want to overwrite. The system symbol table can be hid easily, so the attacker wants to overwrite the IDT table which does not set read-only in Linux.

In 2013, Kees Cook, an Ubuntu Security and Chrome OS security engineer, submitted a patch, x86: Use a read-only IDT alias on all CPUs [6], for the arbitrary memory overwrite vulnerability. Although this patch which Kees Cook submitted prevents attacker from overwriting the IDT table, but we assume that it existed a compatible problem. Because we are not sure that the IDT table will never be modified by legal behavior from the driver of a device; thus, we proposed a adaptive solution on the Kees Cook's patch.

In our solution, we tried to provide a mechanism between being read-only and writable. It is also based on the mechanism of read-only IDT table. We do not care about if the Linux kernel exists any the vulnerabilities of arbitrary memory overwrite. This main idea is similar to the idea of RGBDroid. We also adopt the response-oriented idea instead of the prevention-oriented idea. Our solution has the following key characteristics:

1) It is a software solution that does not require any cost of hardware.
2) It is a compromised solution between being read-only and writable to enhance the compatibility.
3) It is a respond-oriented security solution to avoid spending CPU resources for monitoring.

2. Mechanisms of Attack and Analysis

An attacker will try to modify the credential structures, if he finds out any vulnerabilities of arbitrary memory overwrite. For instance, CVE-2013-2094 is one of classic exploits of arbitrary memory overwrite and its' behaviors can be separated to 3 steps as below:

Step1: The exploit trigger the vulnerabilities by an integer conversion issue.

Step2: It hijacked the exception in the IDT table, so the function pointer of entry of the IDT table was pointed to attacker's privileged code which modified the credential structures. The privileged code was prepared by the attacker in user space.

Step3: The exploit just trigger the exception which is as just mentioned, such as asm volatile ("int $0x4"); and then the exploit will gain the root by modifying the credential structures and restoring the original exception. Finally it opens a shell with root.

At the second step, the exploit was used to writing to known addresses (the IDT table) through the vulnerability of arbitrary memory overwrite; however, it also can modify other kernel function pointer that was found in system.map or proc/kallsyms.

Looking for the kernel function pointer:

1) From disk

```
$ grep do_fork /boot/System.map-3.0.42-0.7-default
ffffffff81058cd0 T do_fork
ffffffff8143959a t do_fork_idle
```

2) From memory

```
$ cat /proc/kallsyms | grep do_fork
ffffffff81058cd0 T do_fork
ffffffff8143959a t do_fork_idle
```

But we also can do some easy defense to protect from above command. Hiding the kernel function pointer:

```
$ chmod o-r /boot/System.map-3.0.42-0.7-default
$ sysctl -w kernel.kptr_restrict=1
```

However, we had hidden the kernel function pointer. Jon Oberheide and Dan Rosenberg proposed Stackjacking technique can leak kernel function pointer by kernel stack [7]. They claimed that the grsecurity/Pax was bypass by Stackjacking.

And then, Spender and Pax team, the authors of grsecurity/Pax, have released some fixes to mitigate the particular exploit vectors.

3. Related Work

3.1. Research on Attacks

The book, A Guide to Kernel Exploitation: Attacking the Core [8], classifies all kinds of Linux kernel vulnerabilities. Although its classification is detailed, the kinds of vulnerabilities are increasing with time.

In May 2013, a famous hacker, sd (sd@fucksheep.org), submitted his exploit for CVE-2013-2094 on the internet. CVE-2013-2094 is a Linux kernel vulnerability for arbitrary memory overwrite. It was caused by sign conversion issues. Although sd's exploit is based on x86-64 architect, the CVE-2013-2094 can be exploited on x86-32 [9] and android [10]. sd's exploit used IDT table to get the root shell; however another exploit by rikiji [9] used the ptmx_fops to get the root shell. There were many malware that can get root shell on Android. In 2014, Vitaly Nikolenko's exploit for CVE-2013-2094 used the default system symbol on Ubuntu 12.04.

3.2. Research on Defence

In 2013, Kees Cook, Google security engineer, submit a patch [6] that set up the attributes of the IDT table read-only while booting. After above patch, Kees Cook published Linux Kernel ASLR [11] on September 19, 2013, in a Linux Security Summit talk.

4. Proposed Solution: Libra

In this section, we want to propose a general solution that protects the arbitrary memory overwrite of privilege escalation. The simple method is to set the IDT table read-only. But the same idea was reported by Kees Cook [6]. He used the _set_fixmap(), Linux kernel API, to put the address of idt_table in the range of fixmap in kernel space and reload it when system was initializing. Therefore, the action of attack read the address of idt_table by SIDT instruction. It will read the read-only idt_table. And then, the action of attack was defended.

4.1. Protection through Paging in x86 Architecture

Linux kernel is used to Paging to manage the virtual memory. Virtual memory is a memory management scheme that allows the execution of processes even without loading them into the main memory. It was also used as a means of memory protection.

When paging in protected mode has been enabled, there are two different types of checks that the processor can perform:

1) User/Supervisor mode checks (facilitated by U/S flag, bit 2)
2) Page type checks (facilitated by R/W flag, bit 1)

The U/S and R/W flags exist both in PDEs and PTEs [12]. _set_fixmap() was implemented by this kind of memory protection, and cleared the R/W flag of PTEs(Page table entries) that pointed to idt_table. When the WP (write protection) flag in CR0 register was enable, the U/S flag and the R/W flag were cleared. Finally, Linux kernel will launch LIDT instruction for reloading the new read-only idt_table.

When exploits try to overwrite the entry of idt table even in supervisor mode. Because the idt table was set to read-only, the CPU will generate the #PF signal to the Kernel.

And then Kernel will jump to do_page_fault() with two parameters [13]:

1) A pointer to a pt_regs structure, which contains the value of microprocessor registers when the page fault occurred.
2) A data type of unsigned long error code that indicates the reason for the page fault.

The do_page_fault() ISR (Interrupt Service Routine) will be invoke when the system reference an invalid memory address.

4.2. A Case Study in CVE-2013-2094 with Read-Only Patch

Here, we use the flow diagram of the page fault handler to analysis of what happened when the system call (perf_event_open) which was hijacked overwrite the entry of idt table. But the idt table was set to Read-only, just as mentioned above.

We take perf_event_open() of CVE-2013-2094 as an example when the idt table was enabled the read-only. When a fault of memory reference occur, the Page Fault Handler will load an invalid address in CR2 register. And then it checks whether the invalid address in kernel space. In this case, the Page fault handler involves the following steps [14] (as shown in Fig. 1):

1) perf_event_open() is a kernel function, so it is in kernel space.
2) When the perf_event_open() was hijacked by the exploit of integer issues, it have been in kernel mode. As a matter of fact, it was implement by checking error_code defined in fault.c.
3) And then, the Page fault handler checks whether the invalid address in vmalloc area. Of course not.
4) It checks the invalid address is a wrong system call parameter. Of course not.
5) In the end, Page fault handler kills the process and kernel OOPS. In summary, we know that the attack with CVE-2013-2094 failed.

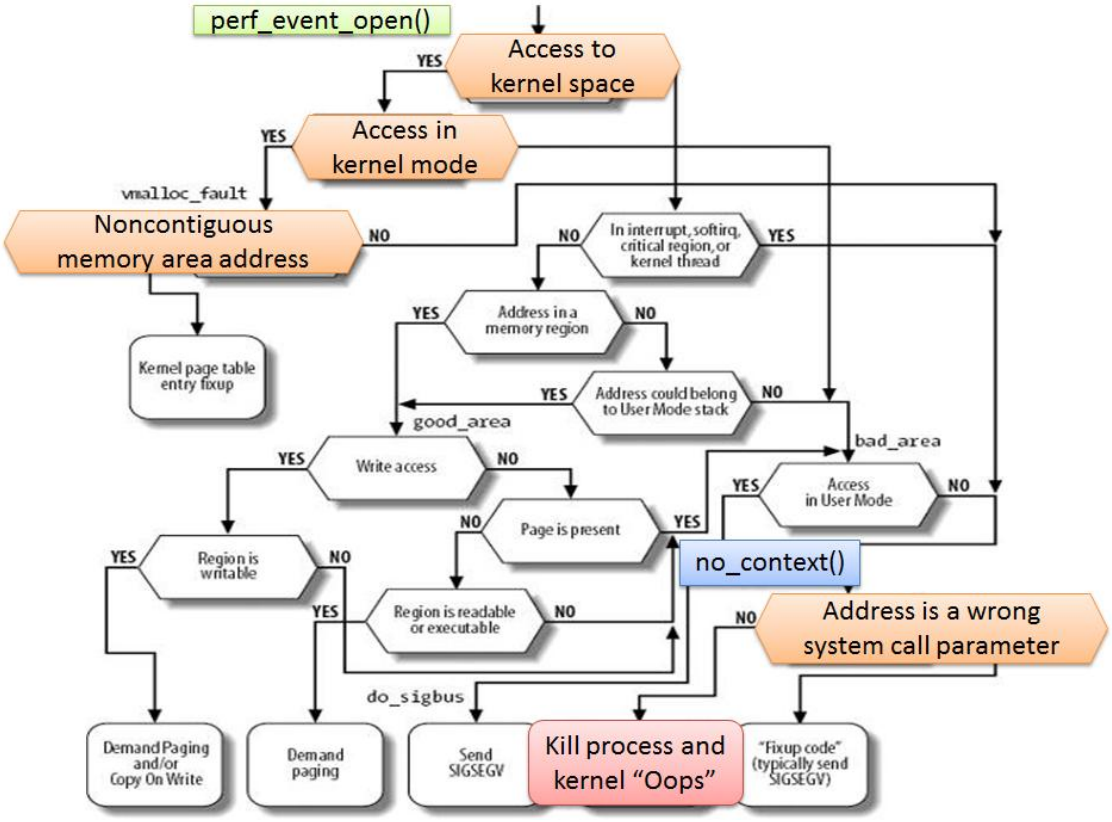

Fig. 1. Trigger the CVE-2013-2094 with read-only patch.

Although the idt_table was set read-only even in supervisor mode, yet it may have had some problem of compatibility. This is, Linux Kernel may need to modify the idt_table, after the Linux System was initializing.

4.3. System Design

In this section, we designed a method for defending against arbitrary memory overwrite. Because we are not sure that the Linux kernel do not modify the entry of idt_table after booting. Therefore, we assume the Linux kernel need to modify the entry of idt_table after booting, and then we try to insert codes into no_context in the page fault handler (/arch/x86/mm/fault.c).

The flowchart of Libra shown in Fig. 2 explained the idea of our design. Because the Linux kernel had kernel space and user space, the Kernel function addresses should be in kernel space that user mode cannot access. If the Kernel function addresses were in user space, it would possibly be modified by attackers even in user mode.

Under the read-only attribute of the IDT table, if any process changes the IDT table with a modified value, it will trigger the page fault handler. Therefore, we create the solution for possible problem of compatibility through the revised page fault handler. Libra is based on this situation. According to the flowchart, we initially check whether the invalid address from read_cr2() is in IDT table or not, that means, from idt_table to idt_table+255×16.

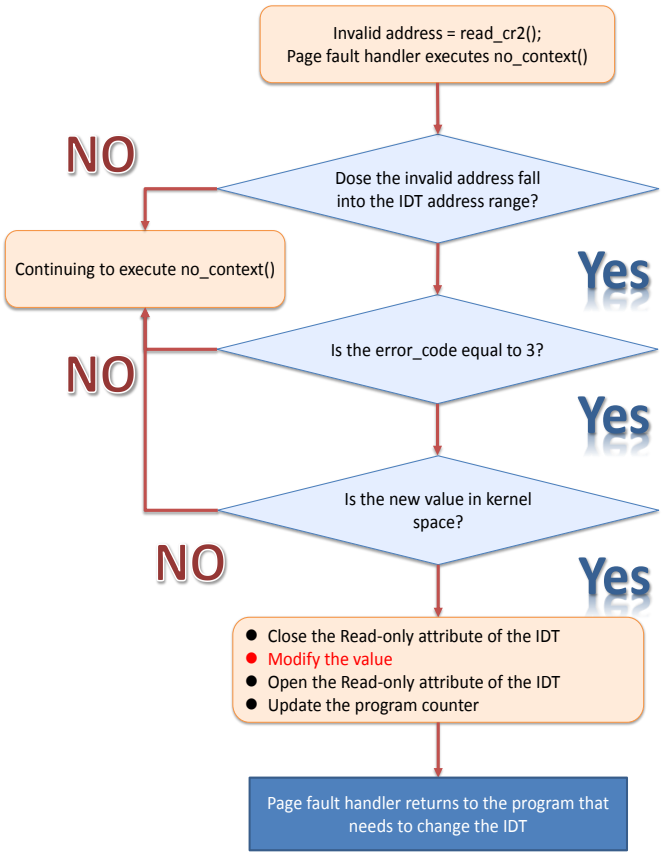

Fig. 2. The flowchart of Libra for IDT table.

And then we check whether the error_code is equal to 3. The error_code is a parameter of do_page_fault(). The source code of Linux kernel defines error_code in fault.c. From the source code and the real-only attribute of the IDT table, we can know that it is a protection fault, because the write access came from a system call by user mode. As mentioned above, the error_code will finally be equal to three statuses shown below:

Page fault error code bits:

bit 0 == 0: no page found

1: protection fault

bit 1 == 0: read access

1: write access

bit 2 == **0: kernel-mode access**

1: user-mode access

In Fig. 2, we can know the adaptive method for the IDT table. In addition, it is also set up to compatible read-only in ptmx_fops data structure. The designed idea of Libra for ptmx_fops is similar to the designed idea of Libra for the IDT table. The flowchart of Libra for ptmx_fops is shown as Fig. 3.

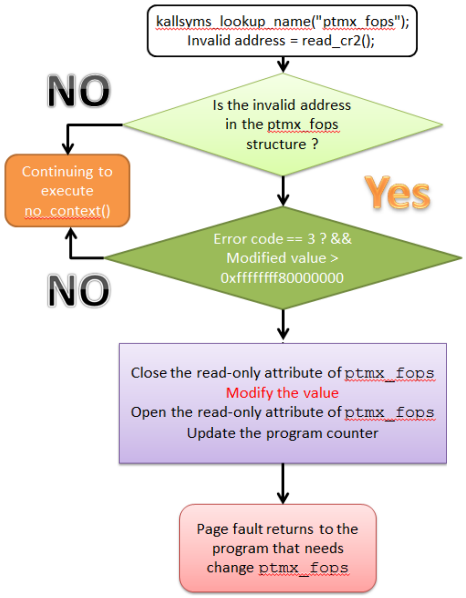

Fig. 3. The flowchart of Libra for ptmx_fops.

5. Evaluation

We implemented our solution on Ubuntu 13.04 (Kernel version: 3.8.8) on x86-64 architecture. Because the solution needs the patch, x86: Use a read-only IDT alias on all CPUs, by Kees Cook. Hence, we upgrade Linux kernel 3.8.8 to Linux kernel 3.10.15. An interrupt handler address in the entry of the IDT table is composed of low_offset, middle_offset and high_offset field [15].

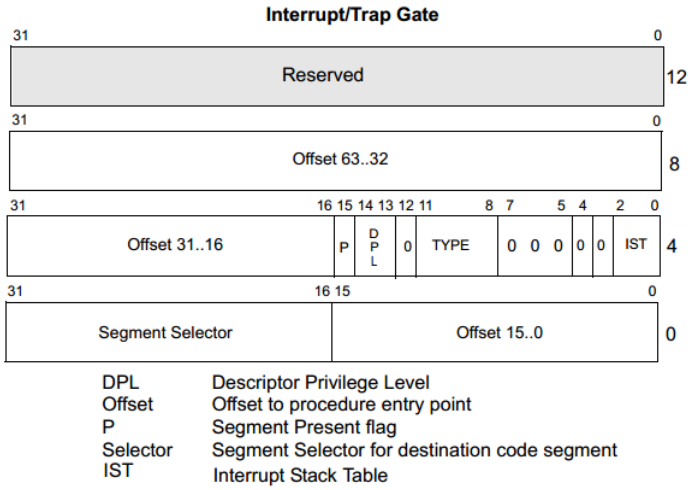

Fig. 4. 64-Bit IDT gate descriptors.

Fig. 4 shows that, in Linux Kernel, The low_offset field is from 0 to 15 bits in offset 0 of Interrupt/Trap Gate. The middle_offset field is from 16 to 31 bits in offset 4 of Interrupt/Trap Gate. The high_offset field is from 0 to 31 bits in offset 8 of Interrupt/Trap Gate [15].

By the flowchart of our design, we check whether the modified value is in kernel space or not. Because the kernel text region is from ffffffff80000000 to ffffffffa0000000 (=512 MB) [16], we know that an address is in kernel space or not by its first 4 bytes roughly. We do not modify the origin value (ffffffff) of high_offset field if users try to change. We consider that the important kernel function pointer needs in kernel space to avoid malicious users' illegal manipulation. Hence, even privileged users cannot modify high_offset field. But low_offset field and middle_offset are allowed to modify by users in our design. This is to say, our design allowed that privileged users can change kernel function pointer in kernel space anywhere. It's an adaptive solution for existent compatible problem possibly in the future.

6. Evaluation

Our adaptive method is based on the Linux patch, X86: Use a read-only IDT alias on all CPUs. That patch puts the address of idt_table to the range of fixmap in traps.c when booting [8]. At the same time, it sets the mapped range of fixmap to read-only. We did not find any existent compatible solution yet. However, we assume a system call that needs to modify the entry of the IDT table after booting.

We installed our adaptive solution on Ubuntu 13.04 64-bit (Kernel version: 3.10.15) with Intel(R) Pentium(R) D CPU 3.00GHz and 1GB RAM to implement all kinds of comparison as below. We deployed the Kees Cook's solution and our adaptive solution on the hardware as mentioned above, respectively.

6.1. Compatibility Comparison

In this section, we design a system call that can overwrite any interrupt handler addresses. Because the exploit for CVE-2013-2094 hijacks the system call (perf_swevent_init), we added a system call (modify_idt) which is similar to the exploit for CVE-2013-2094.

An interrupt handler address is composed of low_offset, middle_offset and high_offset field. The system call which we design will manipulate an interrupt handler address by overwriting the three fields that combine to the interrupt handler address. The system call (modify_idt) will overwrite low_offset, middle_offset and high_offset field as below:

```
idt_table2 = ((gate_desc *) idtr.address);
idt_table2[i].offset_low = 0xbeef;
idt_table2[i].offset_middle = 0xdead;
idt_table2[i].offset_high = 0x12345678;
```

We also programmed an interface (test_modify_idt) for the system call (modify_idt) which we designed. In addition, it is convenient to show the contents of the IDT table; therefore, we programmed a Linux kernel module (showidt_64.ko) to show them. Comparison results between Kees Cook's solution and our adaptive one are shown as Table 1.

Table 1. Kees Cook's Solution V.S. Libra

	Kees Cook's solution	Libra
offset_low	Read-only	Writable
offset_middle	Read-only	Writable
offset_high	Read-only	Read-only

From Table 1, it implies that Libra is more compatible than Kees Cook's solution. Libra allows users to overwrite the kernel space address into kernel function pointer. Otherwise Libra does not allow users to overwrite the user space address into kernel function pointer. Because developers put the important kernel

function pointer in user space, it is not different from that we used in DOS.

6.2. Performance

In this section, we took the performance testing through perf [17] that is in Linux kernel source code and change with the version of Linux kernel. We type the command "perf stat –r 100000 ./test_modify_idt 0" to run the interface 100,000 times for overwriting the zeroth entry of the IDT table.

We executed the above command 100,000 times on the Kees Cook's solution and our adaptive one, respectively. The results showed that Kees Cook's solution takes 132 million CPU cycles and needs about 61 milliseconds of time, and our adaptive solution only takes 116 million CPU cycles and about 59 milliseconds to finish the above commands.

7. Conclusions

The studies in this paper showed the details about attack mechanisms. For example, A case study: CVE-2013-2094 on x86-32. And then, a lot of hackers submitted their patches for Linux kernel at the first moment. However, we proposed a new approach for analyzing the contents of kernel pointer fell in kernel space or user space. If the content of kernel pointer fell in kernel space, the kernel will allow user to change the content; otherwise the kernel will not.

Security is an important issue in the future, but it may make trouble for developers and end-users. Therefore, we tried to implement an adaptive solution satisfying security and compatibility. Libra, an adaptive solution, is pure software solution, and does not need extra hardware for monitoring.

In Libra, we just protect the IDT table and the ptmx_fops structures. But it is not enough. There are about 80,000 system symbols in the Linux kernel symbol table. If we want to fully protect the Linux kernel from this kind of attacks, we have to find out more base addresses of kernel function pointers possibly hijacked by attackers and set them to compatible read-only.

Acknowledgment

The research work was supported by the Ministry of Science and Technology, Government of Taiwan, R.O.C. under The National Project of Taiwan No. 103-2221-E-008-087 and No. 103-2623-E-008-003-D.

References

[1] Jiang, X. Security alert: New sophisticated android malware droidkungfu found in alternative Chinese app market. From http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html

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

[3] Lookout Mobile Security. (2011). *Lookout Mobile Security Technical Tear Down – DroidDream*, Lookout, Inc.

[4] Yeongung, P. (2012). RGBDroid: A Novel Response-based Approach to Android Privilege Escalation Attacks. *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats*. Leet '12, San Jose, CA.

[5] Perf_event_open exploit AT&T ICS Galaxy Note I717. From http://pastebin.com/TLK9Qrgi

[6] Cook, K. x86: Use a read-only IDT alias on all CPUs. From http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=4eefbe792baedb474e256d35370849992fcf1c79

[7] Oberheide, J., & Rosenberg, D. (2011). Stackjacking: Your way to grsecurity/PaX Bypass. *Proceedings of Hackito Ergo Sum*. Paris.

[8] Perla, E., Oldani, M., & Speake, G. (2010). *A Guide to Kernel Exploitation: Attacking the Core, Syngress*.

[9] Rikiji, CVE-2013-2094 port to x86. From http://rikiji.it/2013/05/10/CVE-2013-2094-x86.html
[10] Hiikezoe. CVE-2013-2094 exploit for Android. From https://github.com/android-rooting-tools/libperf_event_exploit
[11] Cook, K. (2013). *Linux Kernel ASLR, Linux Security Summit*, New Orleans.
[12] Blunden, B. (2009). *The Rootkit Arsenal: Escape and Evasion*, 75, Jones & Bartlett Learning.
[13] Gatliff, W. (2001). *The Linux Kernel's Memory Management Unit API*, 3.
[14] Zhang, X. C. [OS] Process address space. From http://www.cnblogs.com/xuczhang/archive/2010/04/02/1703357.html
[15] Intel® 64 and IA-32 architectures software developer's manual volume 3 (3A, 3B & 3C): System programming guide. (2011). *Intel Corp.*, *3A*, 6-17.
[16] The entry in IDT table of definition in Linux Kernel. From ttp://code.woboq.org/linux/linux/arch/x86/include/asm/desc_defs.h.html
[17] Tutorial: Linux kernel profiling with perf. From https://perf.wiki.kernel.org/index.php/Tutorial

Chun-Yi Wang received the M.S. degree in electrical engineering from Chung Cheng Institute of Technology, National Defense University, Taoyuan, Taiwan, in 2006. He pursues a career as a large-scale system programming engineer from 2007 to 2012. He is currently working towards the Ph.D. degree in the Department of Computer Science and Information Engineering from National Central University, Taoyuan, Taiwan. He is interested in networking and system security, especially on honeypot design.

Chieh-Wei Huang received the B.S. degree in the Department of Information Engineering from I-Shou University, Kaohsiung, Taiwan, in 2012. He is currently working towards the M.S. degree in the Department of Computer Science and Information Engineering from National Central University, Taoyuan, Taiwan. He is interested in networking and system security, especially on buffer overflow attacks.

Fu-Hau Hsu received his Ph.D. degree in computer science from Stony Brook University, New York, USA in 2004. He is an associate professor in the Department of Computer Science and Information Engineering at National Central University, Taiwan, R.O.C. His research interests include system security, mobile device security, web security, information hiding, operating system, and networking.

Shih-Jen Chen received the M.S. degree in computer science from National Yunlin University of Science and Technology, Taiwan, in 2000. He is a section manager in the CyberTrust Technology Institute at Institute for Information Industry, Taiwan, R.O.C. His research interests include security compliance checking, mobile device security, web security, IoT security and 4G LTE network security.

Yao-Hsin Chen received the Ph.D. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan. He is currently an engineer at the Information and Communications Research Laboratories, Industrial Technology Research Institude, Hsinchu, Taiwan. His research interests include information security and smart grid.