KERNEL ROOTKIT

ATTACCHI E CONTROMISURE

Matteo Falsetti [aka FuSyS]

Webb.it 02

Padova, 7 Luglio 2002

- La presentazione è basata sul kernel Linux e sulle relative implementazioni delle tecniche di attacco e difesa
- Non sostituisce in alcun modo lo studio diretto del codice sorgente del suddetto kernel e NON è un HOWTO per la compromissione di un sistema Linux/GNU
- È data per scontata una minima conoscenza del linguaggio di programmazione C e del funzionamento interno del kernel Linux

Kernel Rootkit, attacchi e contromisure

- rootkit, nozioni di base
- LKM e rootkit
  - tecniche ed implementazioni
  - contromisure possibili
- Patch del kernel a runtime

rootkit, nozioni di base

- Un rootkit è un insieme di cavalli di Troia che permettono ad un attaccante un successivo ingresso come superutente, nascondendone al contempo ogni attività.

- Spesso l'installazione del rootkit cancella ogni traccia dell'avvenuta compromissione e della sostituzione dei normali binari di sistema.

rootkit, nozioni di base

Bersagli tipici di un rootkit sono:

- ifconfig, netstat
- ls
- ps
- who, w, finger
- login, tcpd, inetd
- ogni possibile demone di rete

rootkit, nozioni di base

Un rootkit contiene spesso:

- Sniffer di rete
- Log-cleaner
- Qualche backdoor

Una semplice ricerca su PacketStorm permette a chiunque di ottenere facilmente svariati rootkit per ogni piattaforma.

rootkit, nozioni di base

- Un semplice metodo per difendersi da un simile rootkit è il confronto delle signature dei binari, con quelle originali del sistema
- Un semplice hash in MD5 o l'uso di software appositi, come Tripwire, permette l'automatizzazione del processo di confronto

LKM e rootkit

Nell'anno 1997,sul numero 50 di Phrack,
esce un articoli di HalfLife:

"Abuse of the Linux Kernel
for Fun and Profit"

- Questo è il primo esempio di implementazione del concetto di
  Hijack delle sys_call

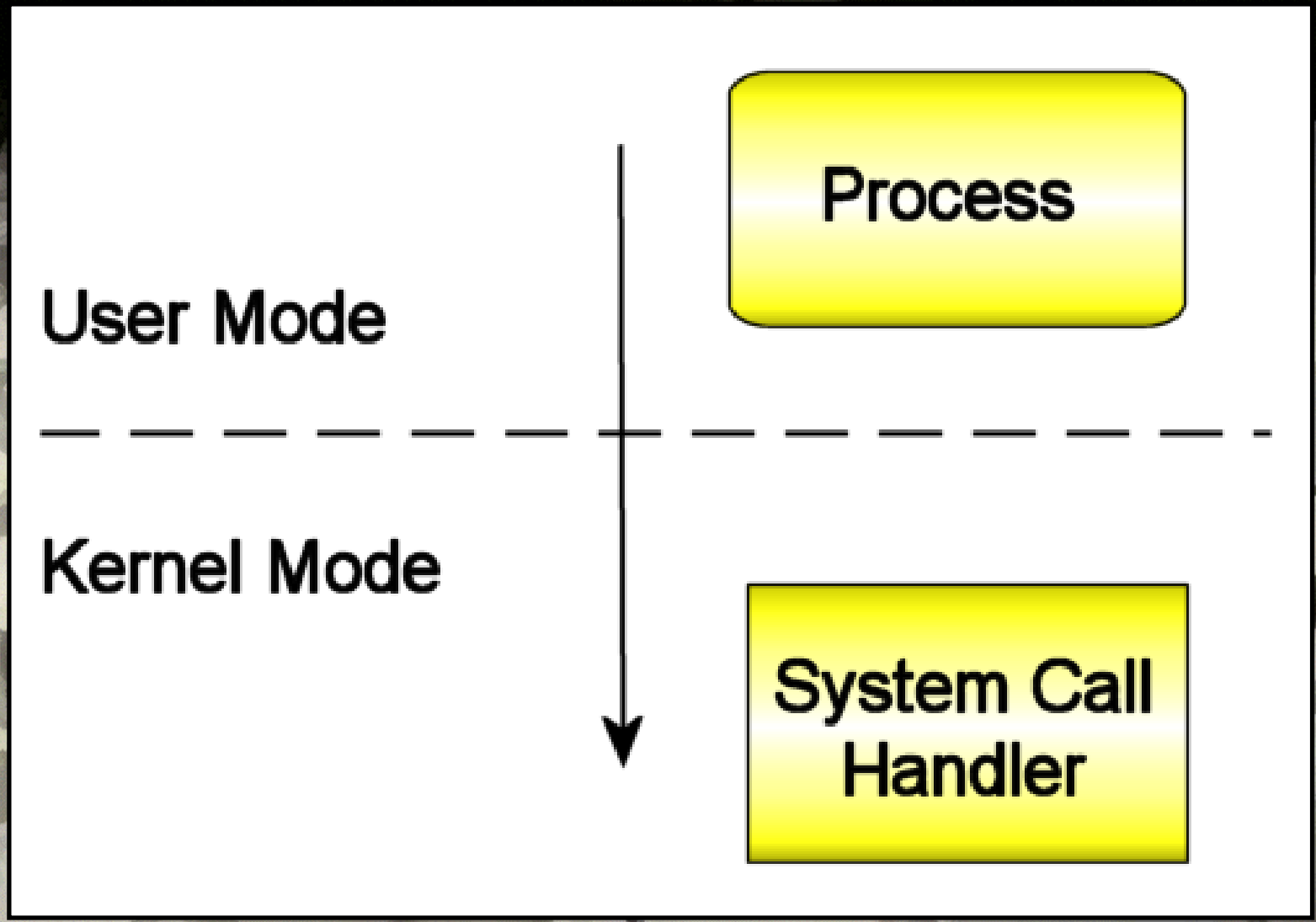
LKM e rootkit

Nel 1998, esce un articolo di plaguez:

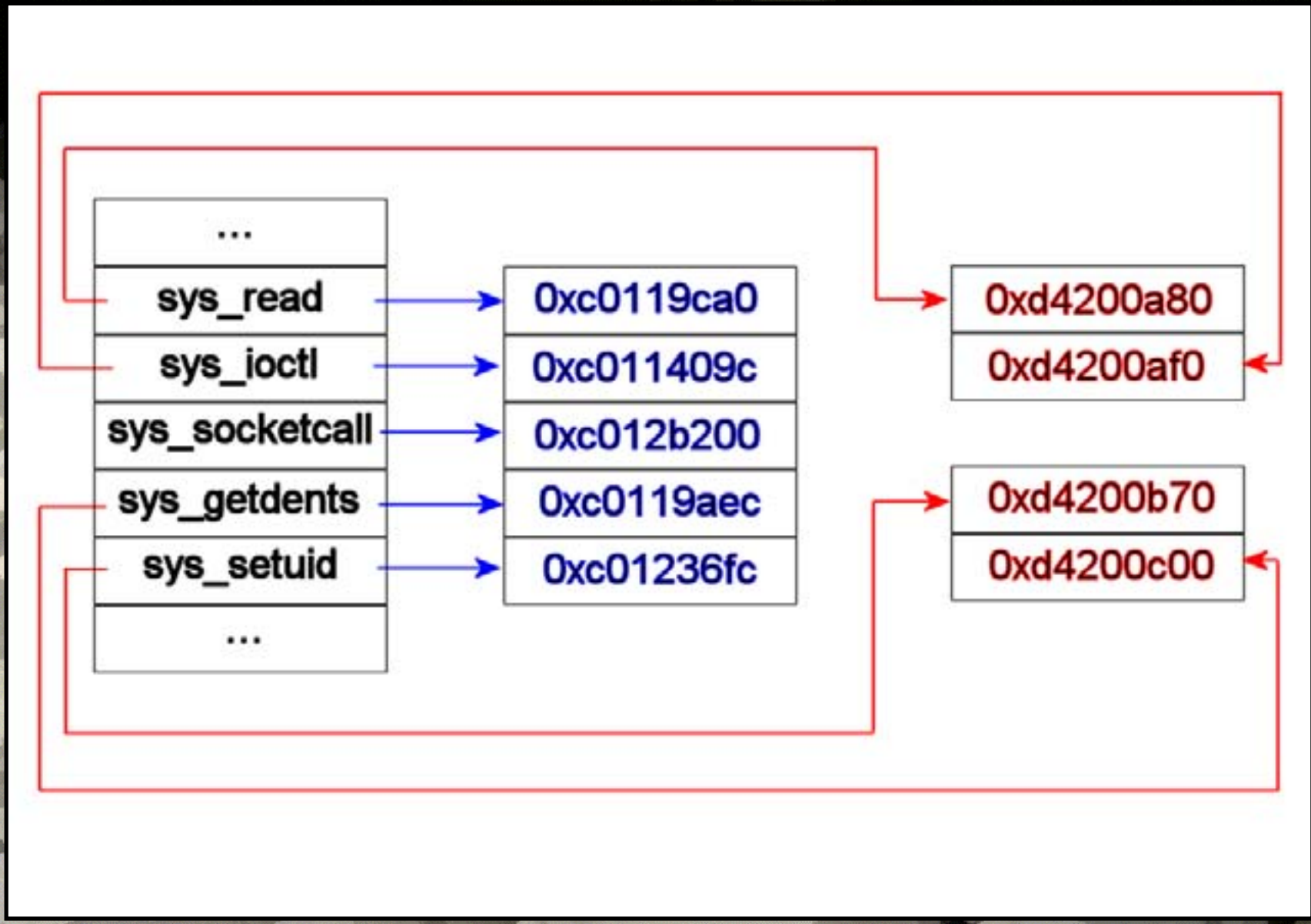
"Weakening the Linux Kernel"

- In esso appare la prima implementazione di un LKM per Linux in grado di sostituire interamente un rootkit in zona utente.
- Questo modulo mostra come effettuare un proficuo hijack delle chiamate di sistema.

LKM e rootkit



LKM e rootkit



LKM e rootkit

- Assistiamo, negli anni successivi, ad una esplosione di sorgenti facilmente reperibili per ogni versione degli UNIX liberi.
- Nel 1999 Pragmatic del gruppo tedesco THC, rilascia un importante documento:

"(nearly) Complete Linux Loadable Kernel Modules"

LKM e rootkit

- itf, heroin, carogna …
- knark, adore, warlkm …

➢ Negli ultimi 4 anni abbiamo assistito ad un sostanziale miglioramento delle tecniche di occultamento di questi moduli, di pari passo con una maggiore incisività di attacco nei confronti del sistema ospite.

Quali sono le sys_call da modificare ?

- Per nascondere files e processi ?
- Per non visualizzare la flag promisc ?
- Per non visualizzare il rootkit LKM ?

In questi casi, strace(1) è un ottimo alleato per l'attaccante…

```
open("/proc/75/stat", O_RDONLY)         = 8
read(8, "75 (cardmgr) S 1 75 75 0 -1 320 "..., 511) = 183
close(8)                                = 0
open("/proc/75/statm", O_RDONLY)        = 8
read(8, "168 168 126 10 0 158 42\n", 511) = 24
close(8)                                = 0
open("/proc/75/status", O_RDONLY)       = 8
read(8, "Name:\tcardmgr\nState:\tS (sleeping"..., 511) = 428
close(8)                                = 0
open("/proc/75/cmdline", O_RDONLY)      = 8
read(8, "/sbin/cardmgr\0", 2047)        = 14
close(8)                                = 0
open("/proc/75/environ", O_RDONLY)      = -1 EACCES (Permission denied)
write(1, "   75 ?        S      0:00 /sbin"..., 41) = 41
stat64(0x40029bb0, 0xbffff50c)          = 0
open("/proc/96/stat", O_RDONLY)         = 8
read(8, "96 (syslogd) S 1 96 96 0 -1 64 2"..., 511) = 177
close(8)                                = 0
open("/proc/96/statm", O_RDONLY)        = 8
read(8, "195 195 168 8 0 187 27\n", 511) = 23
close(8)                                = 0
open("/proc/96/status", O_RDONLY)       = 8
read(8, "Name:\tsyslogd\nState:\tS (sleeping"..., 511) = 428
close(8)                                = 0
open("/proc/96/cmdline", O_RDONLY)      = 8
read(8, "/usr/sbin/syslogd\0", 2047)    = 18
close(8)                                = 0
open("/proc/96/environ", O_RDONLY)      = -1 EACCES (Permission denied)
write(1, "   96 ?        S      0:00 /usr/"..., 45) = 45
stat64(0x40029bb0, 0xbffff50c)          = 0
open("/proc/99/stat", O_RDONLY)         = 8
read(8, "99 (klogd) S 1 99 99 0 -1 320 19"..., 511) = 182
close(8)                                = 0
open("/proc/99/statm", O_RDONLY)        = 8
read(8, "306 306 110 6 0 300 196\n", 511) = 24
close(8)                                = 0
open("/proc/99/status", O_RDONLY)       = 8
read(8, "Name:\tklogd\nState:\tS (sleeping)\n"..., 511) = 426
close(8)                                = 0
open("/proc/99/cmdline", O_RDONLY)      = 8
read(8, "/usr/sbin/klogd\0-c\0003\0", 2047) = 21
close(8)                                = 0
open("/proc/99/environ", O_RDONLY)      = -1 EACCES (Permission denied)
write(1, "   99 ?        S      0:00 /usr/"..., 48) = 48
stat64(0x40029bb0, 0xbffff50c)          = 0
open("/proc/102/stat", O_RDONLY)        = 8
```

```
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 5
access("/proc/net/if_inet6", R_OK)      = 0
socket(PF_INET6, SOCK_DGRAM, 0)         = 6
access("/proc/net/ax25", R_OK)          = -1 ENOENT (No such file or directory)
access("/proc/net/nr", R_OK)            = -1 ENOENT (No such file or directory)
access("/proc/net/ipx", R_OK)           = -1 ENOENT (No such file or directory)
access("/proc/net/appletalk", R_OK)     = -1 ENOENT (No such file or directory)
access("/proc/net/x25", R_OK)           = -1 ENOENT (No such file or directory)
open("/proc/net/dev", O_RDONLY)         = 7
fstat64(0x7, 0xbffff35c)                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40015000
read(7, "Inter-|   Receive                "..., 4096) = 1063
close(7)                                = 0
munmap(0x40015000, 4096)                = 0
ioctl(6, SIOCGIFFLAGS, 0xbffff78c)      = 0
ioctl(6, SIOCGIFHWADDR, 0xbffff78c)     = 0
ioctl(6, SIOCGIFMETRIC, 0xbffff78c)     = 0
ioctl(6, SIOCGIFMTU, 0xbffff78c)        = 0
ioctl(6, SIOCGIFMAP, 0xbffff78c)        = 0
ioctl(6, SIOCGIFMAP, 0xbffff78c)        = 0
ioctl(6, 0x8942, 0xbffff78c)            = 0
ioctl(5, SIOCGIFADDR, 0xbffff78c)       = 0
ioctl(5, SIOCGIFDSTADDR, 0xbffff78c)    = 0
ioctl(5, SIOCGIFBRDADDR, 0xbffff78c)    = 0
ioctl(5, SIOCGIFNETMASK, 0xbffff78c)    = 0
fstat64(0x1, 0xbfffef4c)                = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40015000
write(1, "eth0      Link encap:Ethernet  H"..., 58) = 58
write(1, "          inet addr:192.168.1.2 "..., 73) = 73
open("/proc/net/if_inet6", O_RDONLY)    = 7
fstat64(0x7, 0xbffff39c)                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
read(7, "00000000000000000000000000000001"..., 4096) = 108
read(7, "", 4096)                       = 0
read(7, "", 4096)                       = 0
close(7)                                = 0
munmap(0x40016000, 4096)                = 0
write(1, "          BROADCAST MULTICAST  M"..., 80) = 80
write(1, "          RX packets:0 errors:0 "..., 61) = 61
write(1, "          TX packets:16 errors:0"..., 64) = 64
write(1, "          collisions:0 txqueuele"..., 39) = 39
write(1, "          RX bytes:0 (0.0 b)  TX"..., 53) = 53
write(1, "          Interrupt:11 Base addr"..., 44) = 44
write(1, "\n", 1)                         = 1
close(6)                                = 0
munmap(0x40015000, 4096)                = 0
```

```
fstat64(0x4, 0xbfffedac)                = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4001e000
old_mmap(NULL, 1170948, PROT_READ|PROT_EXEC, MAP_PRIVATE, 4, 0) = 0x4001f000
mprotect(0x40133000, 4052, PROT_NONE)   = 0
old_mmap(0x40133000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 4, 0x113000) = 0x40133000
old_mmap(0x40139000, 15876, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x40139000
close(4)                                = 0
munmap(0x40015000, 34248)               = 0
fstat64(0x1, 0xbfffe12c)                = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40015000
write(1, "Module                  Size  Us"..., 38) = 38
SYS_199(0x4013ee4, 0, 0x40138c00, 0x4013dcf0, 0xbfffe94c) = 0
query_module(NULL, 0, NULL, 0)          = 0
brk(0)                                  = 0x8068e20
brk(0x8068f38)                          = 0x8068f38
brk(0x8069000)                          = 0x8069000
query_module(NULL, QM_MODULES, { /* 8 entries */ }, 8) = 0
brk(0x806a000)                          = 0x806a000
query_module("airo_cs", QM_INFO, {address=0xd5e42000, size=3440, flags=MOD_RUNNING, usecount=0}, 16) = 0
query_module("airo_cs", QM_REFS, { /* 0 entries */ }, 0) = 0
query_module("airo", QM_INFO, {address=0xd5e37000, size=37152, flags=MOD_RUNNING|MOD_VISITED|MOD_USED_ONCE, usecount=1}, 16) = 0
query_module("airo", QM_REFS, { /* 1 entries */ }, 1) = 0
query_module("r128", QM_INFO, {address=0xd5e1d000, size=91712, flags=MOD_RUNNING|MOD_VISITED|MOD_USED_ONCE, usecount=0}, 16) = 0
query_module("r128", QM_REFS, { /* 0 entries */ }, 0) = 0
query_module("agpgart", QM_INFO, {address=0xd5dff000, size=26016, flags=MOD_RUNNING|MOD_VISITED|MOD_USED_ONCE, usecount=1}, 16)
= 0
query_module("agpgart", QM_REFS, { /* 0 entries */ }, 0) = 0
query_module("ds", QM_INFO, {address=0xd5dfc000, size=6480, flags=MOD_RUNNING|MOD_VISITED|MOD_USED_ONCE, usecount=1}, 16) = 0
query_module("ds", QM_REFS, { /* 1 entries */ }, 1) = 0
query_module("yenta_socket", QM_INFO, {address=0xd5df6000, size=8400, flags=MOD_RUNNING|MOD_VISITED|MOD_USED_ONCE, usecount=1},
16) = 0
query_module("yenta_socket", QM_REFS, { /* 0 entries */ }, 0) = 0
query_module("pcmcia_core", QM_INFO, {address=0xd5deb000, size=38624, flags=MOD_RUNNING|MOD_USED_ONCE, usecount=0}, 16) = 0
query_module("pcmcia_core", QM_REFS, { /* 3 entries */ }, 3) = 0
query_module("loop", QM_INFO, {address=0xd58cc000, size=40784, flags=MOD_RUNNING|MOD_AUTOCLEAN|MOD_VISITED|MOD_USED_ONCE, usecou
nt=6}, 16) = 0
query_module("loop", QM_REFS, { /* 0 entries */ }, 0) = 0
write(1, "airo_cs                 3440   0"..., 43) = 43
write(1, "airo                   37152   1"..., 44) = 44
write(1, "r128                    91712   0"..., 34) = 34
write(1, "agpgart                 26016   1"..., 34) = 34
write(1, "ds                       6480   1"..., 44) = 44
write(1, "yenta_socket             8400   1"..., 34) = 34
write(1, "pcmcia_core             38624   0"..., 60) = 60
write(1, "loop                    40784   6"..., 46) = 46
munmap(0x40015000, 4096)                = 0
```

LKM e rootkit

```
extern void *sys_call_table[];
int (*o_getdents) (uint, struct dirent *, uint);

o_getdents=sys_call_table[SYS_getdents];
sys_call_table[SYS_getdents]=(void*)n_getdents;
```

- **n_getdents è la nuova funzione per eliminare ogni riferimento a files e processi dell'attaccante**

Come identificare un LKM “nascosto” ?

- Essenzialmente è possibile operare dall’ interno del kernel, oppure dalla zona utente.
- Il problema con l’approccio modulare risiede nella possibilità che il modulo ‘maligno’ cancelli ogni evenienza di log di difesa e successivamente precluda ogni possibilità di controllo ai moduli ‘di guardia’.

LKM e rootkit

- L’approccio in zona utente rispecchia il funzionamento di tool quali Tripwire.
- Controllando nell’array dei puntatori alle chiamate di sistema i singoli indirizzi, è possibile identificare le operazioni di hijack.
- Due anni fa, la prima versione di kstat(1) ha implementato questo tipo di tecnica.

Nuove Tecniche di Occultamento

- patch della sys_call e NON del puntatore
- creazione di un nuovo array di puntatori alle sys_call, senza modificarne il simbolo esportato
- utilizzo delle file_ops e inode_ops del filesystem /proc invece delle usuali sys_call

patch della sys_call e NON del puntatore

- In questo caso è necessario controllare il codice della chiamata di sistema e confrontarlo con un hash o con una porzione della chiamata.
- Così facendo, chiamate modificate, ma con indirizzo normale, possono essere identificate

nuovo array di puntatori alle sys_call

- In questo caso il semplice controllo dei puntatori non è più sufficiente. Infatti la maggior parte dei tool di controllo riceve il simbolo dell'array delle chiamate attraverso la funzione query_module(). Tale simbolo non viene modificato: viene sostituito invece l'indirizzo dell'array nella funzione system_call() del kernel.

LKM e rootkit

```
void *hacked_sys_call_table;
hacked_sys_call_table=kmalloc(256*sizeof(long int),
    GFP_KERNEL);
memcpy(hacked_sys_call_table, sys_call_table,
    256*sizeof(long int));

(int)*((int*)ptr) =(int) hacked_sys_call_table;
```

- dove ptr punta all'indirizzo originale della tabella delle chiamate nella funzione system_call()

LKM e rootkit

- In questo caso è necessario controllare il codice della chiamata, oppure trovare l'indirizzo della nuova tabella di chiamate attraverso la funzione system_call() oppure scansionando la memoria del kernel alla ricerca di puntatori noti di chiamate non dirottate.

utilizzo delle file_ops e inode_ops di /proc

- Le chiamate di sistema risultano inalterate. Il controllo è comunque banale, dovendo portare il confronto degli indirizzi di memoria dalle sys_call alle file_ops e inode_ops.

```
old_readdir_root = proc_root.FILE_OPS->readdir;
old_lookup_root = proc_root.INODE_OPS->lookup;
proc_root.FILE_OPS->readdir = &new_readdir_root;
proc_root.INODE_OPS->lookup = &new_lookup_root;
```

LKM e rootkit

- Un tipico metodo utilizzato per nascondere i moduli a lsmod(1) consiste nell'estrarre il modulo dalla linked list di struct module presente nel kernel.
- Per far questo sono sufficienti poche istruzioni presenti nella init_module(): è necessario collegare il modulo precedente a quello 'maligno' ad un altro modulo successivamente linkato, nascondendo così il codice in oggetto.

LKM e rootkit

- La nuova versione di kstat(1) per kernel 2.4.x presenta nuove funzionalità che permettono di controllare le sys_call, le operazioni di file ed inode in /proc, le connessioni di rete…
- È inoltre possibile ripristinare la normale tabella delle chiamate ed eseguire uno scan della memoria del kernel alla ricerca di strutture non linkate, per poi poterle rimuovere dopo un patch della linked list.

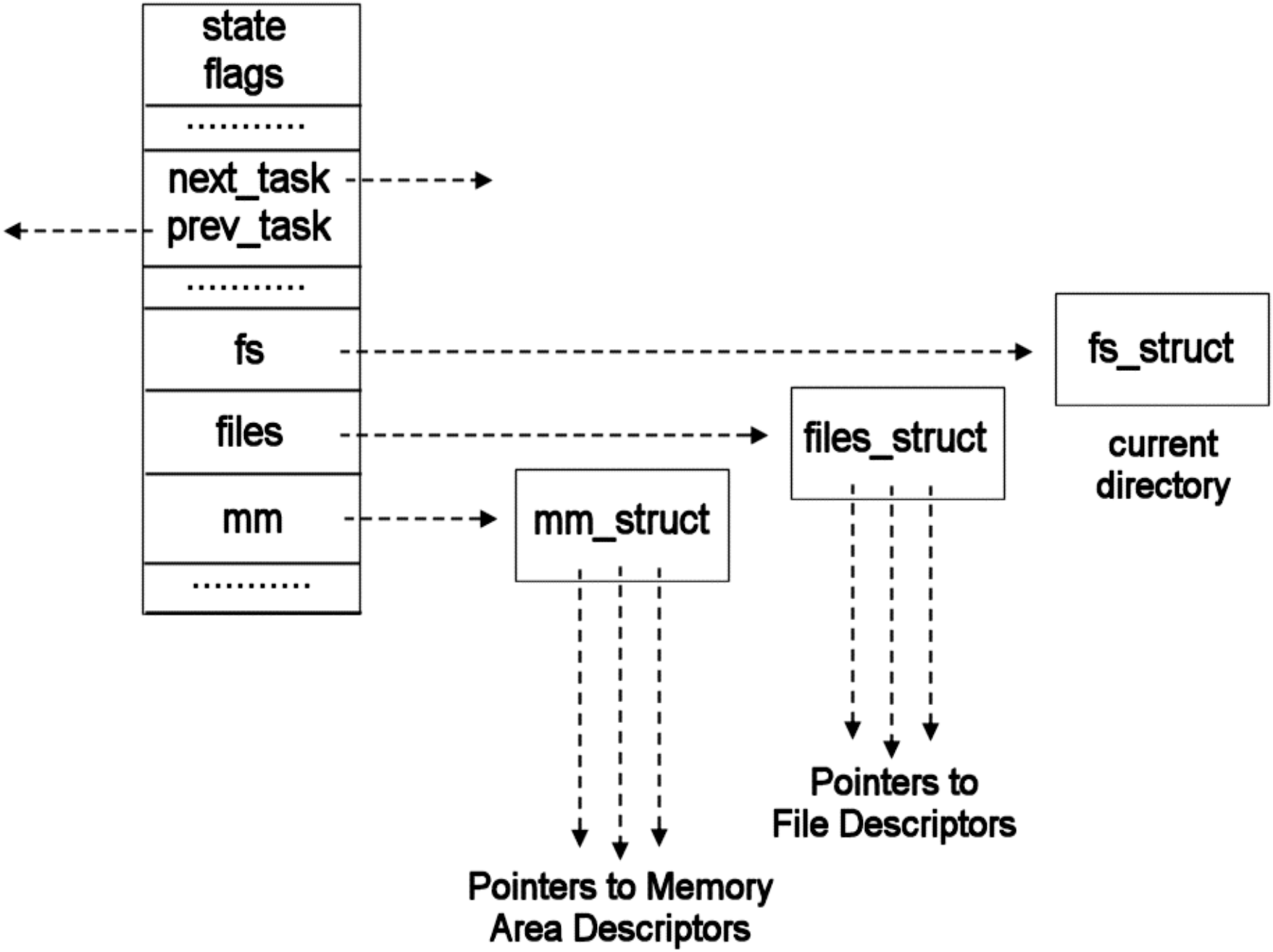
LKM e rootkit

- Il funzionamento base di kstat(1) si basa su /dev/kmem. Attraverso questo file è possibile leggere e scrivere direttamente la memoria del kernel.
- Attraverso /dev/kmem è possibile raggiungere ogni frammento di informazione relativo alle strutture interne del kernel, permettendo al sistemista un controllo granulare, e all'attaccante un efficace patch del sistema on-the-fly.

Patch del Sistema via /dev/kmem

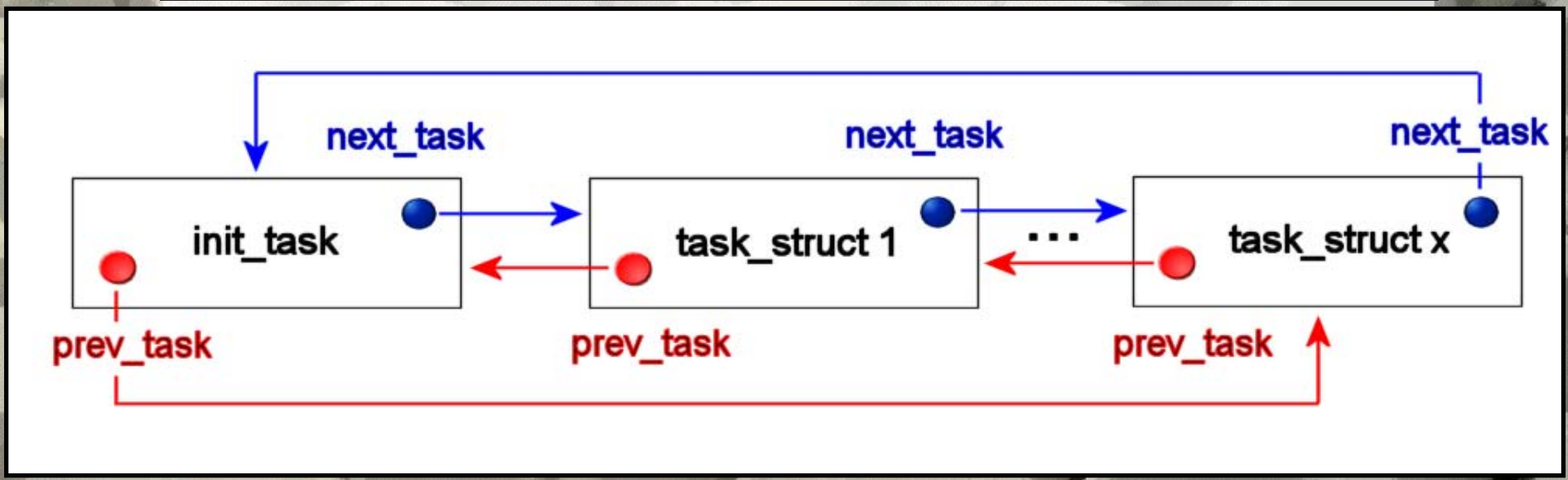
Disabilitare il supporto LKM nel kernel
NON È SUFFICIENTE !

- Un documento di Silvio Cesare ed un articolo uscito nel numero 58 di Phrack presentano facili implementazioni di tecniche per modificare il kernel a runtime. È così possibile inserire codice, come fosse un LKM, direttamente via /dev/kmem. Inoltre è possibile operare direttamente sulle varie strutture del kernel (Es.: task_struct e VFS, …).


state
flags
.....
next_task
prev_task
.....
fs
files
mm
.....
fs_struct
current
directory
files_struct
mm_struct
Pointers to
File Descriptors
Pointers to Memory
Area Descriptors

Patch del Sistema via /dev/kmem

- Le strutture task_struct sono inserite in una double linked list che permette interessanti manipolazioni…



Patch del Sistema via /dev/kmem

- Esempio di controllo su parametri della struct net_device:

```
… leggi da kmem la struct interessata …
  {
    if(dev.flags & IFF_PROMISC)
       dev.flags &= ~ IFF_PROMISC;
    if(dev.gflags & IFF_PROMISC)
       dev.gflags &= ~ IFF_PROMISC;
    if(dev.promiscuity)
       dev.promiscuity = 0;
  }
… scrivi in kmem …
```

Patch del Sistema via /dev/kmem

- Altre possibili modifiche potrebbero interessare le inode_ops di file e directory o funzioni come la get_info() di qualche file in /proc …
- È possibile allocare memoria nel kernel dove inserire codice, funzioni o addirittura dati e files
- Possono essere nascosti processi ed alterati i normali flussi di runtime del kernel

Patch del Sistema via /dev/kmem

- Tutto ciò che è richiesto ad un attaccante, è la conoscenza delle strutture e delle funzioni del kernel.

Come proteggersi da patch via kmem ?

- È necessario controllare l'accesso in scrittura a /dev/kmem. Per far questo esistono le due usuali strade: via kernel e da zona utente.

Patch del Sistema via /dev/kmem

- L’approccio nel kernel richiede la presenza di codice apposito compilato staticamente o linkato come LKM.
- L’approccio da zona utente consiste nell’utilizzo delle Capabilities POSIX 1.e, eliminando la capability CAP_SYS_RAWIO.

kread, funzione di lettura in /dev/kmem

```
int kread(int des, unsigned long addr, void *buf, int len)
{
        int rlen;

        if( lseek(des, (off_t)addr, SEEK_SET) == -1)
                return -1;
        if( (rlen = read(des, buf, len)) != len)
                return -1;

        return rlen;
}
```

Patch del Sistema via /dev/kmem

kwrite, funzione di scrittura in /dev/kmem

```
int kwrite(int des, unsigned long addr, void *buf, int
    len)
{
        int rlen;

        if( lseek(des, (off_t)addr, SEEK_SET) == -1)
                return -1;
        if( (rlen = write(des, buf, len)) != len)
                return -1;

        return rlen;
}
```

Riferimenti e Bibliografia

- "**Abuse of the Linux Kernel for Fun and Profit**", Phrack 50
- "**Weakening the Linux Kernel**", Phrack 52
- "**Sub proc_root Quando Sumus (Advances in Kernel Hacking)**", Phrack 58
- "**Linux on-the-fly kernel patching without LKM**", Phrack 58

  http://www.phrack.com/
- "**Indetectable Linux Kernel Modules**" by SpaceWalker
- **KSTAT** by FuSyS

  http://www.s0ftpj.org/
- "**(nearly) Complete Linux Loadable Kernel Modules**" by Pragmatic

  http://www.thehackerschoice.com/
- "**Linux capability bounding set weakness**", Bugtraq

  Message-ID:

  <Pine.LNX.4.10.10006262044240.1191-100000@s-reynolds.cs.duke.edu>

Riferimenti e Bibliografia (2)

- “**Runtime Kernel KMEM Patching**” by S.Cesare
  http://www.big.net.au/~silvio/runtime-kernel-kmem-patching.txt
- **StMichael**, by Tim Lawless
  http://sourceforge.net/projects/stjude
- “**Understanding the Linux Kernel**”, Bovet & Cesati
  O’Reilly 2001 – ISBN 0-596-00002-2
- **Itf** LKM by plaguez
- **Heroin** LKM
- **oMBRa** LKM by FuSyS
- **KNARK** LKM
- **Adore** LKM by Teso

Contatti

M.Falsetti aka **FuSyS** [SPJ|BFi]

- E-mail: fusys@s0ftpj.org
  fusys@sikurezza.org
- WWW: http://www.s0ftpj.org/