Matteo Falsetti <fusys@sikurezza.org>

Come Muoversi Quando root è Cieco ?

Linux LKM: (in)sicurezza pratica

- 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

- Inserendo la stringa “linux rootkit“ in un motore di ricerca, come Google, otteniamo:

  Risultati 1-10 di circa 1.140

- Specificando una richiesta come ‘ linux AND rootkit ‘ otteniamo invece:

  Risultati 1-10 di circa 16.300

- Un’analoga ricerca su Usenet mostra qualche migliaio di messaggi rilevanti…

- È possibile considerare la possibilità di subire una intrusione informatica come una certezza, assumendo un lasso di tempo infinito.
- Un attaccante esperto o uno script kid ben equipaggiato non mostrano incautamente la loro presenza.
- Le tecniche di occultamento di un intruso che abbia ottenuto id utente == 0 sono oggi raffinatissime rispetto a solo 10 anni fa.

Dal semplice nascondiglio,

.../.evil.c

Dal mascheramento mediante cavallo di Troia,

```
if(strstr(name, secret) != NULL) continue;
```

All'occultamento vero e proprio,

```
if(strstr((char *)&(dirptr->dname),
 (char *)SUBVISUS) != NULL) {...}
```

root non è sicuro neanche del suo kernel...

Cronologia Essenziale delle Tecniche Pubbliche di Abuso del Kernel

HalfLife, 1997, Phrack #50

Plaguez, 1998, Phrack #52

Pragmatic, 1999, LKM Hacking Guide

Palmers, 2001, Phrack #58

Sd & Devik, 2001, Phrack #58

SpaceWalker, 2001, Indetectable Linux LKM

Palmers, 2002, Phrack #59

Cronologia Essenziale delle Tecniche Pubbliche di Abuso del Kernel (2)

FuSyS, 1998, BFi #3

FuSyS, 1998, BFi #4

pIGpEN, 2000, BFi #8

FuSyS, 2000, BFi #8

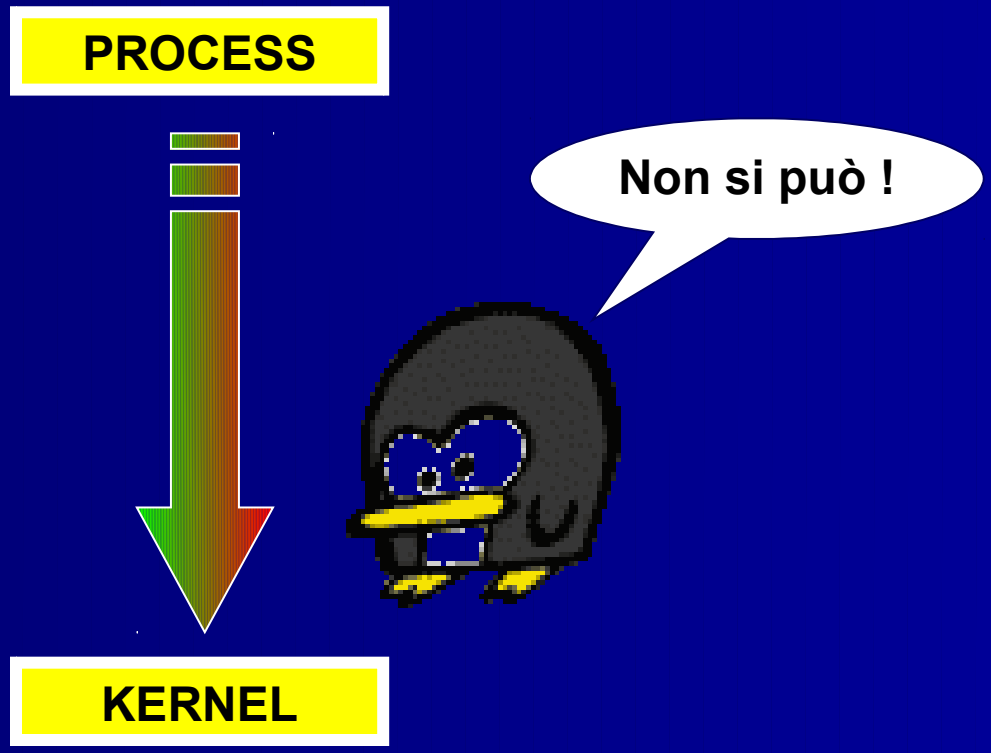
Vecna, 2002, BFi #11

Xenion, 2002, BFi #11

Dark-Angel, 2002, BFi #11

Concetto basilare dell'abuso del kernel è uno:

Dirottamento delle sys_call

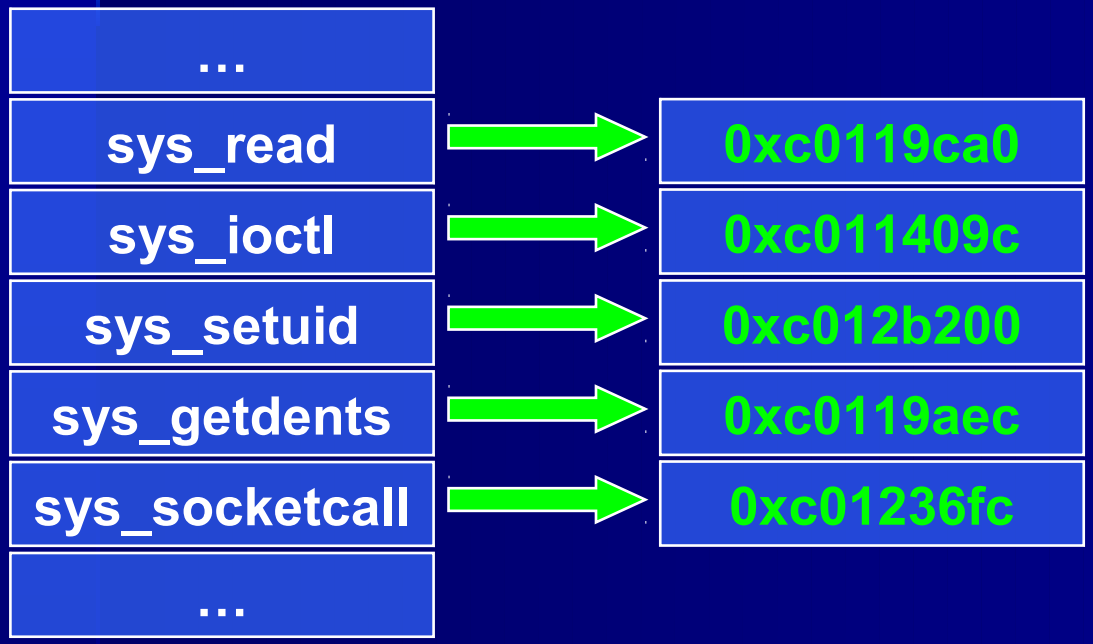


Concetto basilare dell'abuso del kernel è uno:

Dirottamento delle sys_call

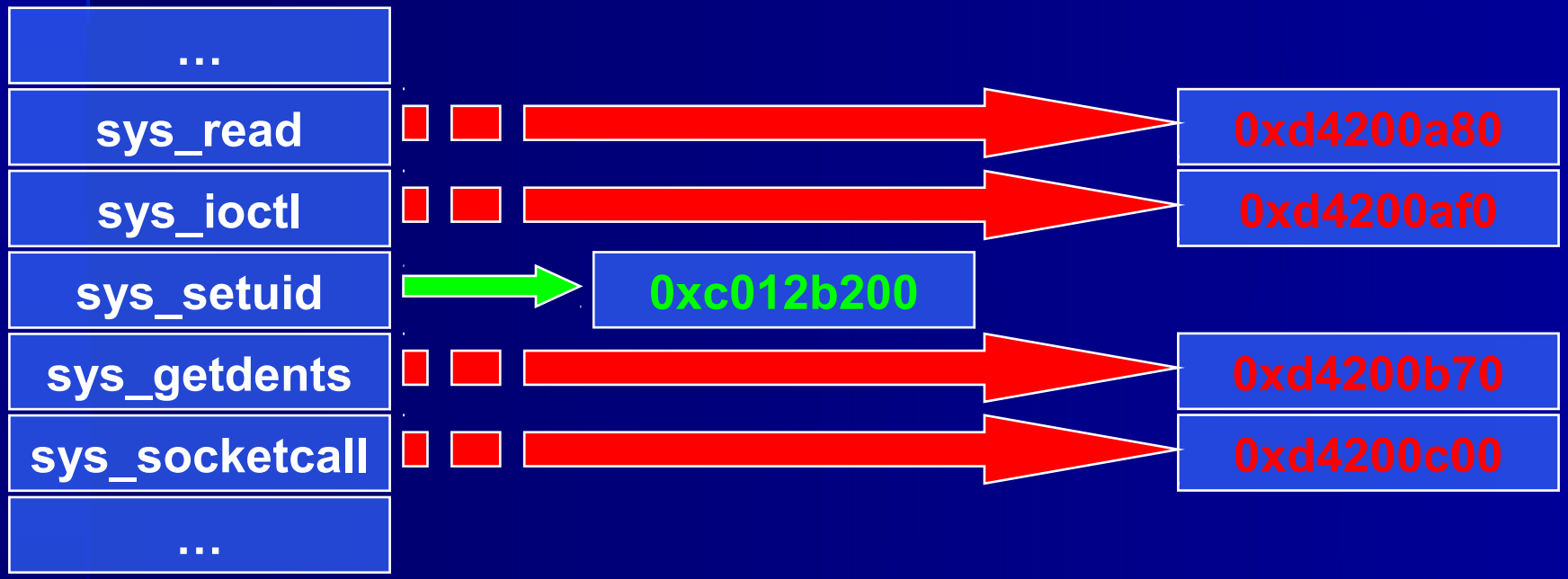
Tecniche di Abuso delle sys_call

Dirottamento delle sys_call



Tecniche di Abuso delle sys_call

Dirottamento delle sys_call



- Un attaccante con le opportune conoscenze può tranquillamente creare un LKM adatto alle sue esigenze e al sistema bersaglio.
- È richiesta una conoscenza specifica del kernel da sovvertire, delle sue funzioni e strutture dati.
- Queste conoscenze sono, nel mondo Open Source, liberamente disponibili; in alternativa è possibile il download di strumenti già pronti da modificare.
- In ogni caso, strace(1) è amico non del solo sistemista o sviluppatore …

```
[fusys@LinuxDay /tmp] $       strace whoami
```

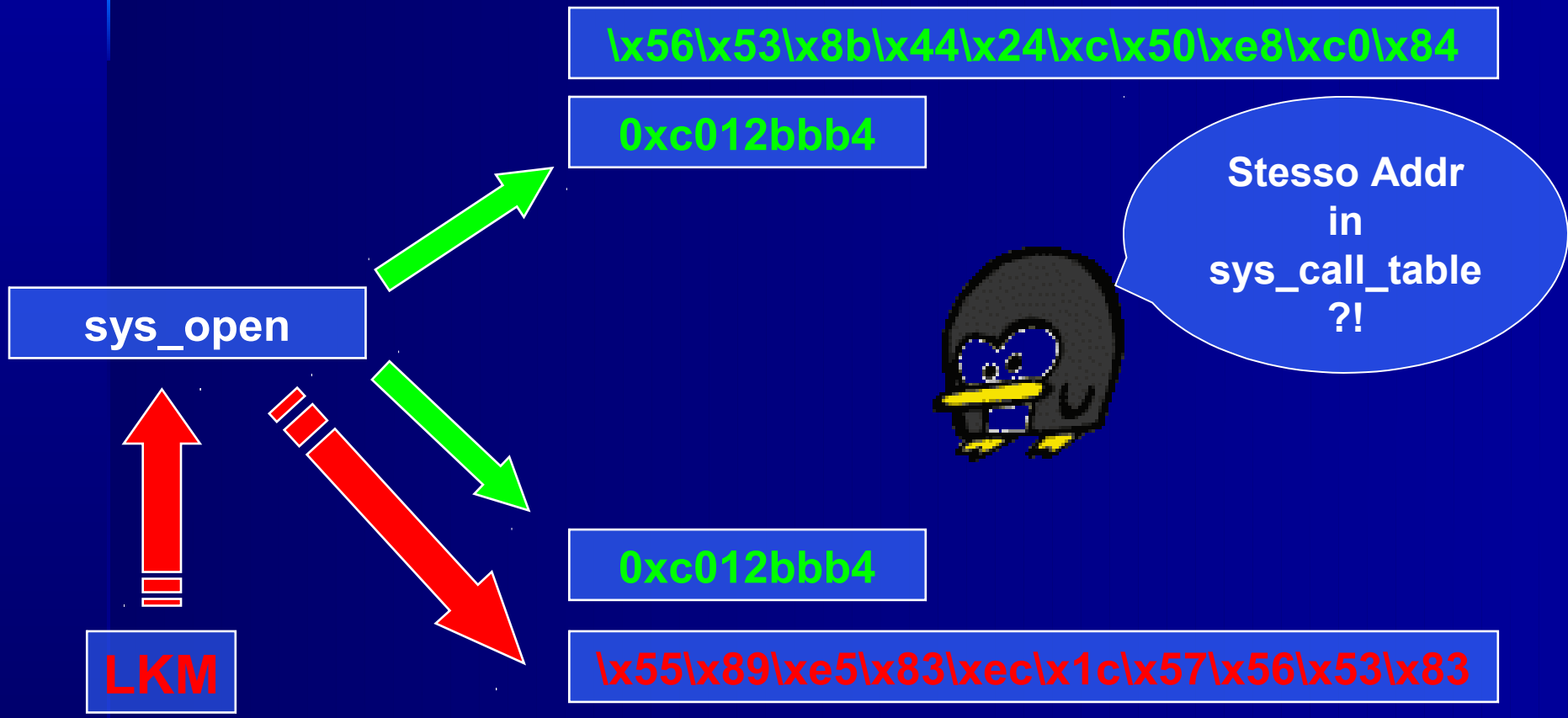
```
execve("/usr/bin/whoami", ["whoami"], [/* 50 vars
*/]) = 0
…
geteuid()                                   = 500
getuid()                                    = 500
getgid()                                    = 100
getegid()                                   = 100
…
open("/etc/passwd", O_RDONLY)                     = 3
…
write(1, “fusys\n”, 6fusys
}                                           = 6
```

```
stat64(0x40029bb0, 0xbffff50c)          = 0
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
```

 # strace ps ax

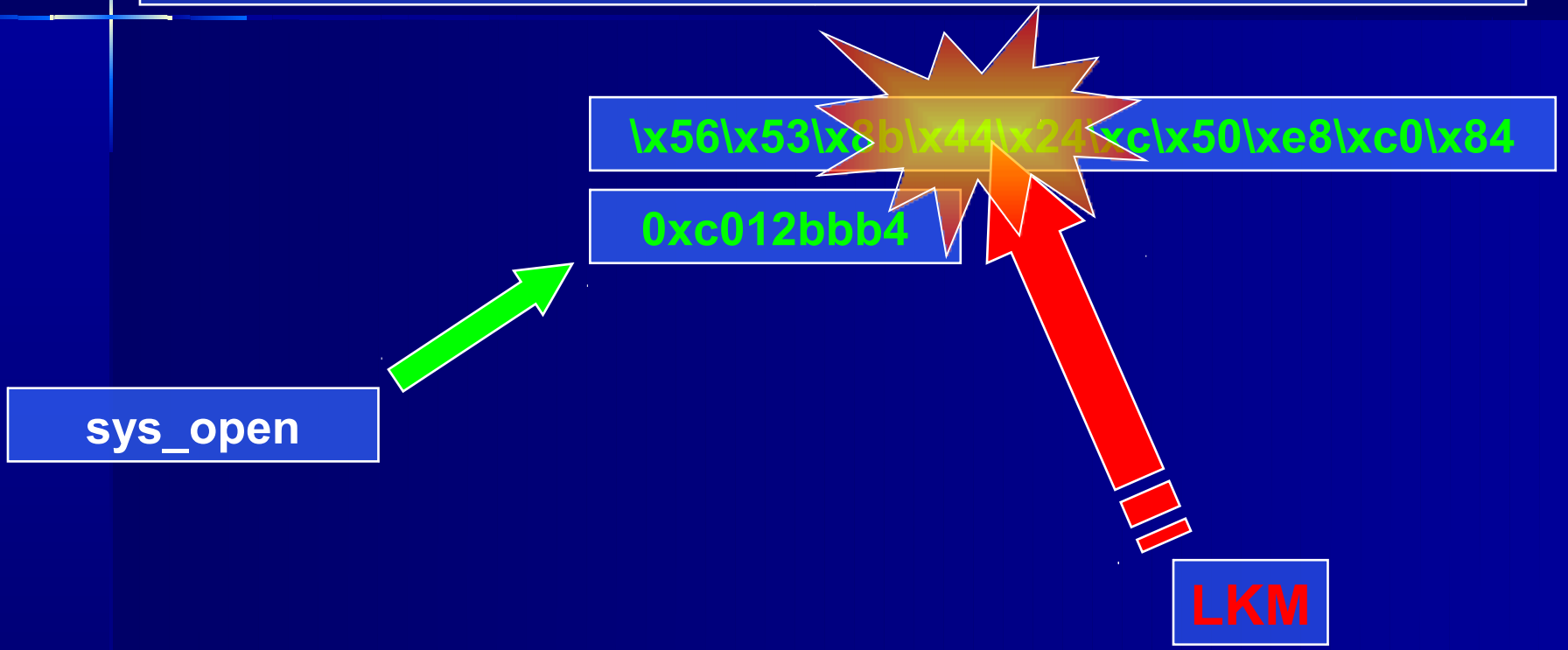
Tecniche Avanzate di Abuso delle sys_call

Modifica delle sys_call (1)



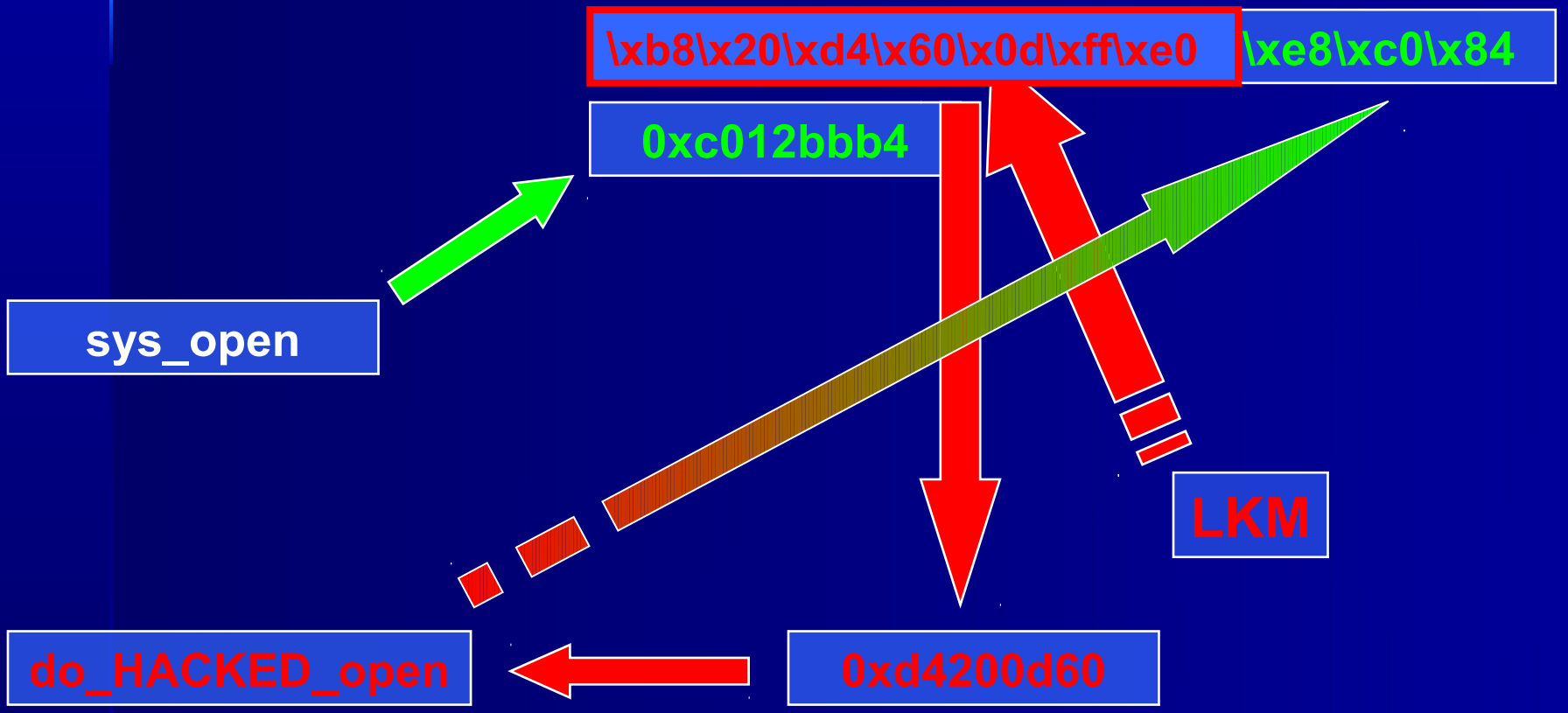
Tecniche Avanzate di Abuso delle sys_call

Modifica delle sys_call (2)



Tecniche Avanzate di Abuso delle sys_call

Modifica delle sys_call (2)



Tecniche Avanzate di Abuso delle sys_call

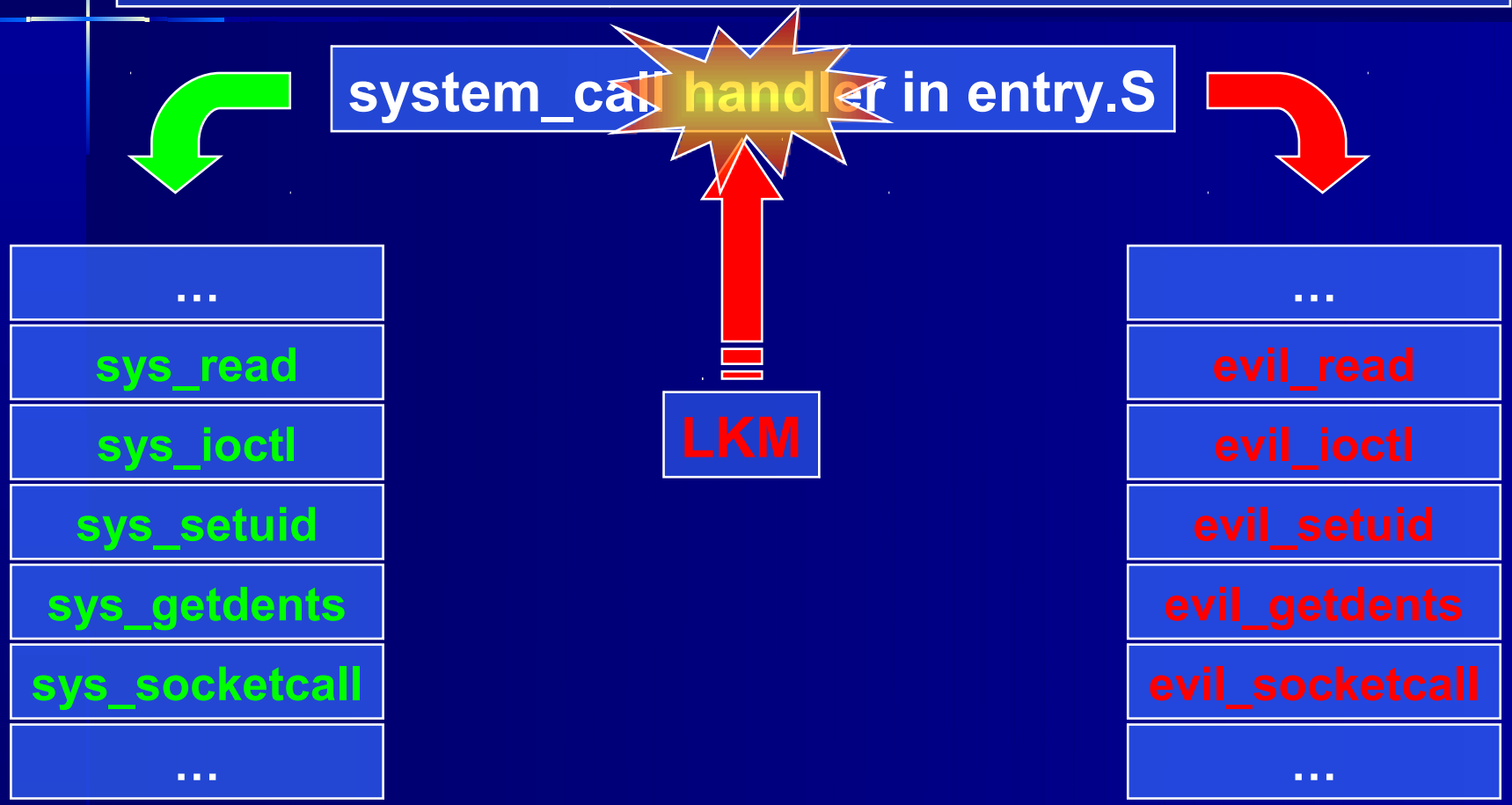
Clonazione Stealth di sys_call_table (1.a)

```
unsigned sys_call_off, sct;
char sc_asm[100], *p;
```

```
asm ("sidt %0" : "=m" (idtr));
readkmem (&idt, idtr.base + 8*0x80, sizeof(idt));
sys_call_off = (idt.off2 << 16) | idt.off1;
readkmem (sc_asm, sys_call_off, 100);
p = (char*)memmem (sc_asm, 100, "\xff\x14\x85", 3);
sct = *(unsigned*)(p+3);
```

Tecniche Avanzate di Abuso delle sys_call

Clonazione Stealth di sys_call_table (1.b)



Tecniche Avanzate di Abuso delle sys_call

Clonazione Stealth di sys_call_table (1.b)

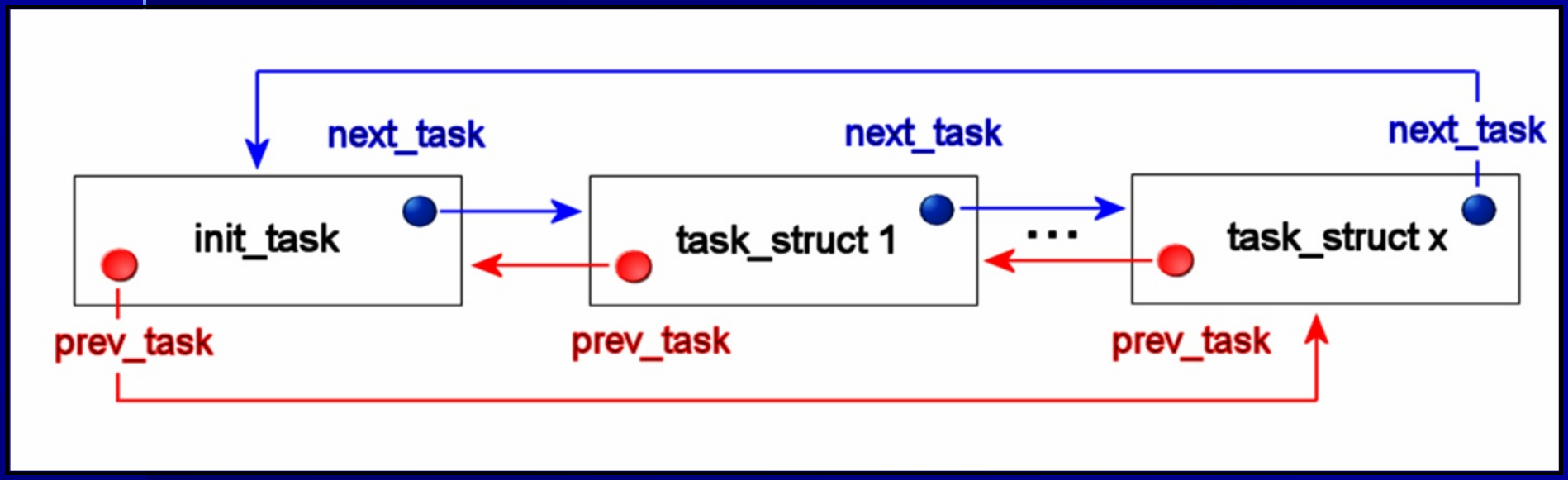
system_call handler in entry.S

- 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.

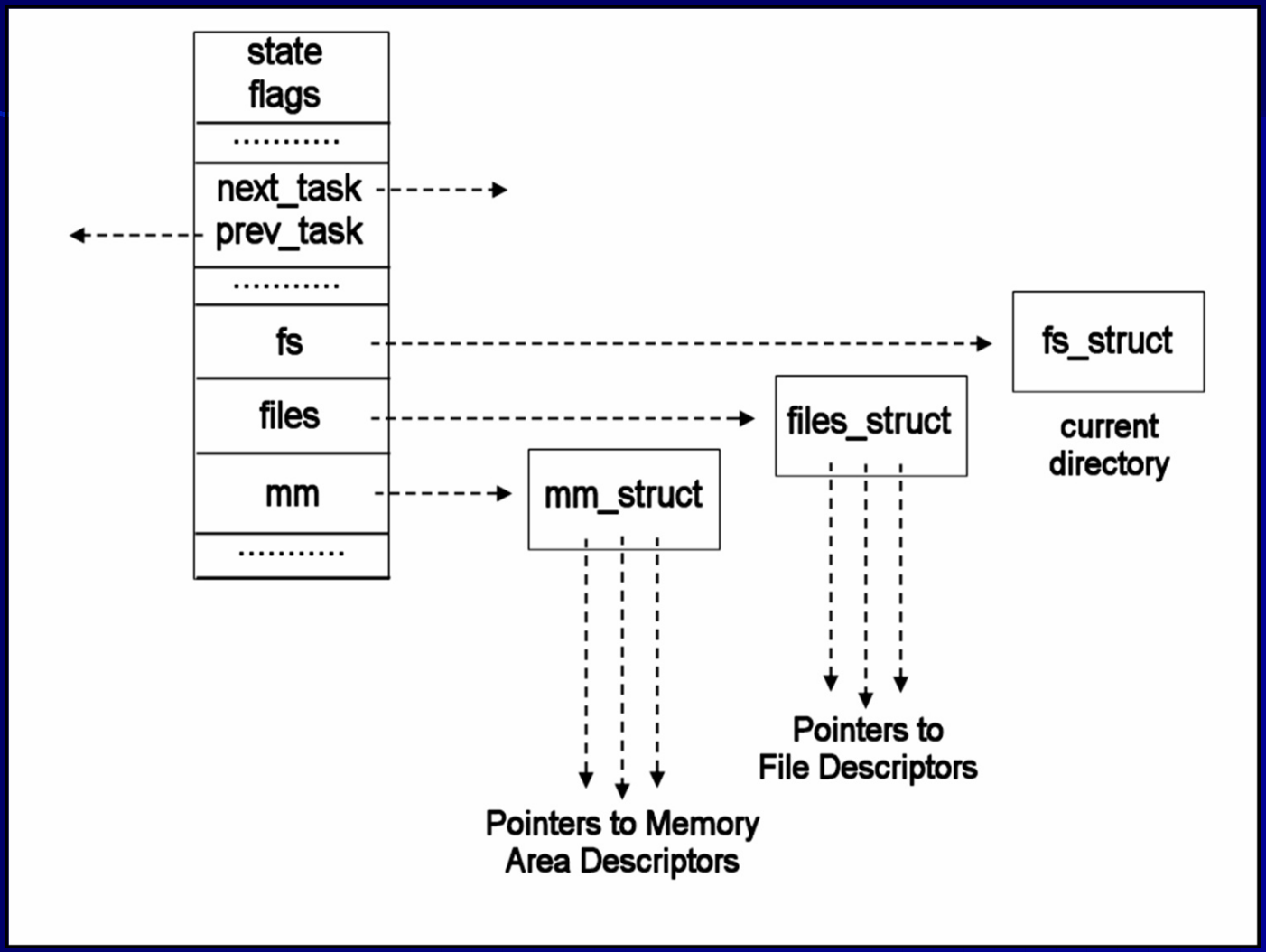
- La nuova versione di **kstat(1)** per kernel 2.4.x presenta nuove funzionalità che permettono di controllare le sys_call, le operazioni di file ed inode in /proc, le connessioni di rete…
- È inoltre possibile ripristinare la normale tabella delle chiamate ed eseguire uno scan della memoria del kernel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

- 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.

Manipolazione Lista dei Processi (1)



Manipolazione Lista dei Processi (2)



Manipolazione Dati dell'Interfaccia di Rete

… leggere da /dev/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;
}
```

… scrivere in /dev/kmem …

Manipolazione Diretta di /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;
}
```

Manipolazione Diretta di /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;
}
```

- È **indispensabile**:
  - ✓ controllare l'handler delle sys_call
  - ✓ controllare la sys_call_table via IDT
  - ✓ effettuare scan delle liste dei processi e LKM
  - ✓ controllare i file descriptor di ogni task_struct
  - ✓ ripristinare on-the-fly strutture e indirizzi
  - ➢ cronometrare l'esecuzione delle sys_call
  - ➢ effettuare scan del pool di memoria disponibile a kmalloc()
  - ➢ ?!

È **importante capire che**:

- ✓ non può esistere il tool perfetto in zona kernel
- ✓ se un attaccante ha accesso privilegiato al kernel, allora è in grado di ottenere ciò che preferisce
- ✓ LKM maligni e k-IDS non possono che scontrarsi in eterno come in una affascinante partita a scacchi senza vincitori netti e/o certi
- ✓ servono livelli ridondanti di sicurezza, come ad esempio le capabilities, LIDS, LSM, (…)
- ✓ spesso l'intuito è importante e decisivo quanto un ottimo IDS, ma ancor più spesso si confonde nella paranoia :-p

- "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
- "Indetectable Linux Kernel Modules", SpaceWalker
- "(nearly) Complete Linux Loadable Kernel Modules", Pragmatic
- 5 Short Stories about execve, Phrack 59
- Kernel Function Hijacking, Silvio Cesare
- Runtime Kernel KMEM Patching, Silvio Cesare
- Progetto Caronte, FuSyS
- oMBRa LKM, FuSyS
- KSEC & KSTAT, pIGpEN, FuSyS
- HKS: Hacking Kernel Structures, Vecna
- Kernel Hacking: Nuove Tecniche per l'Occultamento, Xenion
- Smashing the Kernel for Fun and Profit, Dark-Angel
- Linux Device Drivers – 2nd ed., Rubini & Corbet
- Understanding the Linux Kernel, Bovet & Cesati

Domande ?

Matteo Falsetti	aka FuSyS
E-mail	m.falsetti@oltrelinux.com fusys@sikurezza.org fusys@s0ftpj.org
WWW	www.s0ftpj.org www.sikurezza.org