Betölthető rendszermagmodulok programozása és rendszerhívás-elfogás

A jelenlegi processzorok két üzemmódban képesek működni: rendszermag és felhasználói üzemmódban. Rendszermag üzemmódban több utasítást használhatunk, ezenkívül az egész memóriát és az összes regisztert elérhetjük. A megszakításvezérlők és az operációs rendszer szolgáltatásai is a rendszermag üzemmódban (röviden magmódban) futnak. Ezzel szemben, ha a processzor felhasználói módban fut, akkor kisebb utasításkészlettel dolgozhatunk, a memóriának és az eszközöknek pedig csak egy részét látja a processzor. Ezt az üzemmódot a könyvtárfüggvények és a felhasználói programok használják. E két üzemmód teremti meg a mai operációs rendszerekben a biztonságot és megbízhatóságot.

A programok legtöbbször felhasználói módban futnak. Magmódba csak különleges, az operációs rendszerrel kapcsolatos feladatok elvégzéséhez váltanak át.

Az operációs rendszer szolgáltatásait rendszerhívásokon keresztül érhetjük el. A rendszerhívások a rendszermaghoz vezető „átjárók”, melyeket programból megvalósított megszakításokkal hoztak létre, az operációs rendszer pedig magmódban kezeli azokat.

Az operációs rendszer rendszerhívási táblázatot tart fönn, ennek mutatói a rendszermagban lévő, a hívásokat végrehajtó rendszerfüggvényekre hivatkoznak. A program számára ez a táblázat teszi elérhetővé az operációs rendszer szolgáltatásait. A különféle rendszerhívások listáját a `/usr/include/sys/syscall.h` fájlban találjuk meg. Linuxban ez a fájl a `/usr/include/bits/syscall.h` fájlt is magában foglalja.

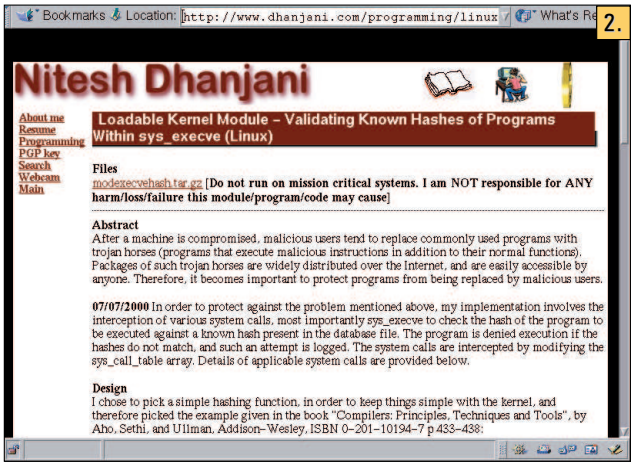
A betölthető modulok olyan kódrészek, melyeket szükség esetén a rendszermagba tölthetünk. Ezek további képességekkel bővítik a magot, anélkül, hogy a gépet újra kellene indítanunk. Linuxban a modulokat gyakran alkalmazzák új eszközmeghajtók elkészítéséhez. A betölthető modulok használata helyett a bővítéseket fordítjuk be közvetlenül a rendszermagba, így mindig egyetlen fájllal dolgozhatunk. Ennek viszont az a hátránya, hogy minden bővítés alkalmával újra kell fordítanunk a magot.

A mag programozása nemcsak rendkívüli összetettsége miatt nehéz, hanem azért is, mert a hibakeresés nagyon sok időt vesz igénybe. Az operációs rendszer hibakereséséhez minden új mag telepítése után újra kell indítanunk a gépet. A fejlesztéshez ezért mindenképpen a betölthető modulokat ajánljuk, hiszen így csak ritkán kell újrafordítanunk a magot, és a gép újraindítására sincsen szükség. A másik fontos ok, hogy mivel a felhasználónak nem kell a létező magot eltávolítania vagy helyettesítenie, sokkal szívesebben veszi a modulokkal megvalósított új lehetőségeket igénybe.

A mag modultámogatásában fontos szerepet játszik a rendszerhívások elfogása, ennek a lehetőségnek az alábbi példákhoz hasonló módon vehetjük hasznát. Megjegyeznénk, hogy a továbbiak megértéséhez szükséges némi C programozói ismeret.

A rendszerhívások

Minden operációs rendszerben a rendszerhívások teszik lehetővé, hogy a felhasználói programok közvetlenül elérjék a mag szolgáltatásait. Különbséget kell tennünk a rendszerhívások (system calls) és a könyvtárfüggvények (library functions) között. A könyvtárfüggvények egy adott programhoz kapcsolódnak és „hordozhatóbbak”, hiszen nem függenek a magtól. Azonban sok könyvtárfüggvény



rendszerhívásokkal hajt végre bizonyos feladatokat a magban. Ennek bemutatására nézzük meg most az alábbi C programot (*pelda1. c)*), mely megnyit egy fájlt és kiírja annak tartalmát:

```
#include <stdio.h>

int main(void)
{
   FILE *myfile
   char tempstring[1024]

   if(!(myfile=fopen("/etc/passwd","r")))
   {
      fprintf(stderr,"Nem lehet megnyiti a
       fájlt\n");
      exit(1);
}

while(!feof(myfile))
{
   fscanf(myfile,"%s\n",tempstring);
```

```
    fprintf(stdout,"%s\n",tempstring);
}

exit(0);
}
```

Ebben a programban az fopen függvénnyel nyitottuk meg a `/etc/passwd` fájlt. Fontos azonban megjegyezni, hogy az fopen nem rendszerhívás. Az fopen az open rendszerhívást hívja meg a tulajdonképpeni I/O művelet elvégzéséhez. Egz program által meghívott rendszerhívások listáját a strace programmal jeleníthetjük meg. Azt feltételezve, hogy a fenti programot `a.out` néven fordítottuk le a gcc-vel, a `strace ./a.out` parancs hatására az `a.out` által használt rendszerhívásokat tekinthetjük meg.
A mag a rendszerhívás segítségével vált át a folyamat gazdájának felhasználói azonosítójára (userid). Ha tehát a fenti programot egy közönséges felhasználó indítaná el, értékként az általa nem olvasható `/etc/shadow` fájlt adva meg, az open nem lenne képes a feladatát végrehajtani, így a fenti if feltétel „igaz” értéket kapna, és a „nem lehet megnyitni a fájlt” hibaüzenet jelenne meg.

Példa a rendszerhívások elfogására a betölthető modulokon keresztül

Tegyük fel, hogy az exit rendszerhívást szeretnénk elfogni, s minden ezt meghívó folyamatnak egy, a konzolra kiírandó üzenetet kívánunk átadni. Ehhez meg kell írnunk saját exit rendszerhívásunkat, majd meg kell adnunk a magnak, hogy az eredeti exit helyett a sajátunkat hívja. A saját exit hívás végén akár az eredeti exitet is meghívhatjuk. Ehhez a rendszerhívási táblázatot (sys_call_table) kell módosítanunk. Nézzük meg a `/usr/src/linux/arch/i386/kernel/entry.S` fájlt (természetesen csak akkor, ha i386-rendszerű gépen dolgozunk). Ez a fájl a mag összes rendszerhívását és a sys_call_table-ben elfoglalt helyüket tartalmazza.
A sys_call_table-t úgy kell módosítanunk, hogy a sys_exit a saját exit hívásunkra mutasson. Az eredeti sys_call-ra mutató hivatkozást mentenünk kell és saját rendszerhívásunk végén, ennek kell átadnunk a vezérlést. A fentieket az *pelda2 c*-ben látható kóddal oldhatjuk meg. A példa programját a `gcc -Wall -DMODULE -D__KERNEL__ -DLINUX -c pelda2.c` paranccsal fordíthatjuk le. Ekkor a pelda2.o nevű modul jön létre, melyet a rendszermagba úgy tölthetünk be, hogy rendszergazdaként kiadjuk az `insmod pelda2.o` parancsot. Most győződjünk meg arról, hogy a konzolon vagyunk-e (hiszen a printk csak a konzolra ír), majd futtassunk egy olyan programot, mely az exit rendszerhívást használja. Például az `ls` parancs kiadására a `„Szia! A sys_exit hívásakor ez volt a hibakód: 0”` üzenet jelenik meg.
Most adjuk ki az `ls` parancsot egy nem létező fájlra, így a program 0-tól különböző hibakóddal lép ki. Tehát az `ls nem_létező_fájl` hatására a `„Szia! A sys_exit hivasakor ez volt a hibakod: 1”` szöveg jelenik meg.
A betöltött modulok listáját az `lsmod` paranccsal tekinthetjük meg. A modul eltávolításához az `rmmod pelda2` parancsot használjuk.

Egy érdekesb példa: a sys_execve elfogása a trójai falovakkal szembeni védelemhez

A trójai falovak olyan programok, melyek valamelyik rendszerszolgáltatásba vagy programba beépülve káros kóddal egészítik ki azokat. Ha egy gépre betörnek, akkor a rossz szándékú behatoló gyakran helyez el trójai falovakat a rendszerben. Ezeket sok helyről le lehet tölteni, tehát mindenképpen meg kell akadályoznunk, hogy kedves ajándékként valaki ezekkel „bővítse” programjainkat.
Következő példánkban is a rendszerhívások elfogásával foglalkozunk: a védelem megvalósításáért a `sys_execve` segítségével

pelda2.c

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <sys/syscall.h>

extern void *sys_call_table[];


asmlinkage int (*eredeti_sys_exit)(int);

asmlinkage int sajat_exit (int error_code)
{


       /*minden meghíváskor kiírja a szöveget a
       konzolra*/
       printk ("Szia! A sys_exit hívásakor ez
       ➥volt a hibakód:  %d\n",error_code);
       /* az eredeti sys_exit-et is meghívjuk*/
       return eredeti_sys_exit(error_code);
}

/* ez a függvény a modul betöltésekor kerül
   meghívásra */
int init_module()
{
       /* mentjük az eredeti sys_exit-re
          mutató hivatkozást */

eredeti_sys_exit=sys_call_table[__NR_exit];

       /* módosítjuk a sys_call_table-t, hogy
          az eredeti sys_exit
          helyett a sajátunkat hívja */

sys_call_table[__NR_exit]=sajat_exit_fuggveny;
}

/* ez a függvény a modul eltávolításakor kerül
   meghívásra */
void cleanup_module()
{
/* a modul eltávolításakor az __NR_exit megint
   az eredeti *sys_exit-re mutat */

sys_call_table[__NR_exit]=eredeti_sys_exit;
}
```

ellenőrizzük, hogy a programhoz tartozó indexelés egyezik-e az előzőleg egy adatbázisban tárolt indexeléssel. Ha a két adat nem egyezik, a program nem indulhat el, és minden ilyen próbálkozásról naplóbejegyzés készül. A védelmet például az alábbi lépésekkel valósíthatjuk meg:

1. Elfogjuk a `sys_execve` hívást, majd a futtatandó fájl kiszámított fájlleíróját (inode) összehasonlítjuk az adatbázisban található értékkel. A fájlleírók a fájlrendszer adatszerkezetei, a fájlokkal kapcsolatos adatokat tartalmazzák. Mivel minden fájlhoz egyedi fájlleíró tartozik, az összehasonlítás eredményéből pontosan meghatározhatjuk a következő lépést. Ha nincsen találat, akkor

az eredeti `sys_execve` meghívása után visszatérhetünk. Ha azonban találunk ilyet, akkor kiszámítjuk a programhoz tartozó indexelést, majd ezt összehasonlítjuk az indexelt adatbázissal. Ha itt is van találat, akkor meghívjuk az eredeti `sys_execve`-t és visszatérünk. Ha nincs találat, akkor a próbálkozásról naplóbejegyzést készítünk, és hibajelzéssel visszatérünk.

2. Elfogjuk a `sys_delete_module` hívást. Ha a modul nevével hívtak bennünket, akkor hibajelzéssel térünk vissza. A modult nem engedjük törölni.
3. Elfogjuk a `sys_create_module` hívást és hibajelzéssel térünk vissza. Az új modulok beillesztését megtiltjuk, hiszen nem szeretnénk, hogy egy rossz szándékú programozó modulja elfogja az 1. lépésben említett `sys_execve` hívást.
4. Elfogjuk a `sys_open` hívást, így megelőzhetjük azt, hogy az indexelt adatbázist vagy a naplófájlt bármilyen program illetéktelenül megnyissa írásra.
5. A `sys_unlink` hívás elfogásával megakadályozzuk az idexelt adatbázis vagy a naplófájl törlését.

Tartsuk szem előtt, hogy a fenti módszer nem jelent teljes körű védelmet; de első nekifutásra nem is rossz. Egy rossz szándékú felhasználó például módosíthatja a `/dev/kmem` magszimbólumait, vagy közvetlen eszközeléréssel írhat a lemezre, s így az open megkerülésével is írhat az indexelt adatbázisba. Vagy, mivel ez a példa egy betölthető modul, a behatoló egyszerűen letilthatja a modul rendszerindításkor történő betöltését a `/etc/rc.d` fájlok módosításával. Mindezek mellett még számos más rendszerhívással módosítható vagy törölhető az indexelt adatbázis vagy a naplófájl.

Ami a legfontosabb: legyünk tisztában azzal, hogy a betölthető modulokat a behatoló saját terveinek végrehajtására is felhasználhatja. Például a `sys_execve` függvényhívás elfogásával trójai falovat, a read és write rendszerhívások elfogásával pedig billentyűzetfigyelő eljárást építhet be a rendszerbe. Behatolás esetén a betölthető modulok rugalmassága és hatékonysága tehát komoly veszélyforrást jelent. A Kapcsolódó címek részben felsorolt honlapokon további példaprogramokat is találhatunk a témával kapcsolatban.

Gustavo Rodriguez-Rivera (grr@cs.purdue.edu) a Purdue Egyetem vendégprofesszora és a Geodesic Systems programmérnöke. Érdeklődési körébe az operációs rendszerek, a hálózat- és memóriakezelés tartozik.

Nitesh Dhanjani (dhanjani@dhanjani.com) a Purdue Egyetem végzős hallgatója. Operációs rendszerekkel, hálózatokkal és a biztonsággal foglalkozik. Több cég, így például az Ernst & Young LLP, számára végzett már biztonsági felméréseket, szabadidejében pedig tanácsadást is vállal.

Kapcsolódó címek

Betölthető magmodulok Linux alatt *(1. kép)*
➲ http://packetstorm.securify.com/groups/thc/LKM_HACKING.html

Linux Kernel Module Programming Guide (Ori Pomerantz)
➲ http://howto.tucows.com/LDP/LDP/lkmpg/

Nitesh Dhanjani munkái *(2. kép)*
➲ http://www.dhanjani.com/programming/linuxexechash/

Várakozás elkerülése minden 21. újraindításnál

Mikor a rendszer az újraindításkor befűzi a lemezrészeket, általában húsz alkalommal kimarad az fsck futtatása, a huszonegyedik újraindításkor viszont az összes fájlrendszert leellenőrzi.
Ez a hosszas várakozás bizony bosszantó lehet. Honnan lehet tudni, hogy hányadik befűzésnél tartasz egy bizonyos fájlrendszer esetében? Írd be:

```
# dumpe2fs /dev/hda7 | grep
          ➥'[mM]ount count'
dumpe2fs 1.19, 13-Jul-2000 for EXT2 FS
0.5b, 95/08/09
Mount count:                7
Maximum mount count:        20
```

Látható, hogy a /dev/hda7 az utolsó fsck óta hét alkalommal lett befűzve, és az fsck húsz alkalommal ugorja át az ellenőrzést.
Ha az összes fájlrendszerednek azonos a befűzés-számlálója (mount count), akkor a rendszer valamennyit egyszerre fogja ellenőrizni.
Ezen könnyű segíteni:

```
# umount /dev/hda6
# tune2fs -C 9 /dev/hda6
tune2fs 1.19, 13-Jul-2000 for EXT2 FS
0.5b, 95/08/09
Setting current mount count to 9
# mount /dev/hda6
```

Így a befűzés-számlálót 9-re állíthatod.
Figyelem! A tune2fs programot kizárólag befűzetlen fájlrendszeren futassuk. A tune2fs akkor is végrehajtja feladatát, ha a fájlrendszer használatban van, de gyanítom, hogy ez veszélyes, szóval tőled függ, óvatos leszel-e.
Tételezzük fel, hogy négy fájlrendszered van. Állítsd be a befűzés-számlálókat a következőképpen: 1,6,11,16. Ezáltal az ellenőrzés egyenletesen oszlik meg közöttük.
A fentiek végrehajtásával a várható újraindítási idő azonos marad, csak az indítási idő egyenetlensége csökkent. Az, hogy kedvelni fogod-e ezt a módszert, attól függ, mennyire vagy türelmetlen, de az eredeti megoldásnál kétségkívül jobb.
Ha igazi rögeszmés vagy és szereted, ha sokszor fut az fsck, vagy ha több mint 20 fájlrendszered van, a legnagyobb befűzési számot is megváltoztathatod a `tune2fs -c N` végrehajtásával. Az `N=-1` érték kikapcsolja az ellenőrzést. Jó, ha tudod azt is, hogy a `tune2fs -i 2` jelentése: „kétnaponta ellenőrizz”. Ez akkor jöhet jól, ha például hordozható számítógépet használsz.