Betölthető magmodulok felhasználási módjai

Védekezzünk a kalózok eszközeivel és tegyük biztonságosabbá a rendszerünket a segítségükkel.

Sereghyi számítógépes biztonsági eszköznek közös a forrása: gépkalózok világa. Az olyan eszközöket, mint a kapupásztázó vagy a jelszófeltörő programok, eredetileg rossz-szándékú emberek tervezték, hogy velük fenyegessék a számítógépes rendszerek biztonságát. Mostanra azonban már a rendszergazdák is felhasználják ezeket kiszolgáló számítógépeik és felhasználói nevük biztonságának átvizsgálására. E cikk egy ilyen gépkalózötletet mutat be – a rendszermagmodul kiaknázását –, és példát ad arra, hogyan fokozható a rendszer biztonsága néhány ugyanilyen módszer és ötlet felhasználásával. Legelőször a saját ötletem történetét és működési elvét fogom felvázolni, ezt követően néhány példa segítségével megkísérlem eloszlatni a rendszermag programozását körüllengő mítoszt. Végül egy ugyancsak hasznos, de terjedelmesebb példát fogok megvizsgálni, amely a rendszert érő támadások egész sorozatát segít elhárítani.
Mielőtt munkához fognánk, szót kell ejtenem még a szabványos lemondási nyilatkozatról. Tudatában kell lenned, hogy a rendszermag memóriaterületén levő programhiba képes tönkretenni a számítógépedet, a rendszermag területén előforduló végtelen ciklus pedig lefagyaszthatja. A fentieknek megfelelően tehát ne fejlessz vagy ne próbálgass programmodulokat a termelésben közvetlenül résztvevő számítógépen, a fejlesztett programmodulokat alaposan vizsgáld át, hogy biztosítsd rendszered üzembiztonságát és adataid épségét. Annak érdekében, hogy a programfejlesztés során a rendszerösszeomlásból eredő adatvesztést a lehető legkisebbre csökkentsük, a program kipróbálásra virtuális gépet vagy más működésmódot utánzó programot használjunk, úgymint Bochsot, plex86-ot, felhasználói üzemmódú Linux-kaput, avagy VMware-t, illetve a munkaállomásra naplózó állományrendszert telepíthetünk, mint például az SGI XFS-rendszere. A fent mondottakon kívül fontos tudni, hogy írásunkban szereplő példák egyike sem lett SMP-gépen kipróbálva, és nagyon is valószínű, hogy többprocesszoros gépeken nem működőképesek. Nos, minthogy a bennünket érintő kérdéseket megtárgyaltuk, térjünk át a modulokra.
Néhány hónapja linuxos belső nyomkövető programon dolgoztam. Minden folyamatra vonatkozóan nyomon szerettem volna követni a rendszerhívásokat az általuk átvett adatokkal együtt. A kísérletezgetés során számos megközelítést kipróbáltam, de egyik sem volt annyira sikeres, mint amennyire szerettem volna. Itt van például mindjárt a `libc` függvény a `write()`-nál, amely lehetővé tette számomra, hogy a C-programokból kezdeményezett `write()` hívásokat naplózzam, de a dinamikus bináris eszközkezelés már csak azokra a végrehajtható állományokra korlátozódott, amelyeket az eszközkezelő könyvtár elemi utasításokra tudott bontani (C, C++ és Fortran). Az, hogy az ellenőrzés csak néhány programnyelv által előállított állomány átvizsgálására korlátozódik, kisebbfajta megszorítás, hiszen a GNU/Linux-rendszeren úgyszólván minden programot C-ben, C++-ban, vagy olyan programnyelven írtak, amely rendelkezik C, illetve C++-alapú futásidejű programkönyvtárral, ilyen a Perl vagy a Python programnyelv. Csakhogy a szóban forgó megoldás eme hiányossága engem elméleti szinten valóban zavart. Tudtam, milyen egyszerűen félre lehet vezetni a rendszert egy kevésbé ismert, nem C-re vagy C++-ra támaszkodó programnyelvből indított rendszerhívással, vagy egy gépi nyelven (assembly) készített. Világos volt, hogy lehetetlen a felhasználói területet átvizsgáló, megkerülhetetlen eszközt készíteni, valamint az is egyértelművé vált, hogy elég nehéz hasznos eszközt készíteni úgy, hogy az ember ne ásná be magát a rendszermag rejtelmeibe. Minthogy nem kívántam rendszerfoltozással foglalkozni, és a fordítás-újraindítás-próbálgatás ciklusa sem volt vonzó a számomra, nem hittem benne, hogy ezt a kérdést a rendszermag területén meg lehet oldani.
Alighogy aggodalmaimat várakozólistára tettem, és dolgozni kezdtem ezen a mostani feladaton, a helyi LUG- (Linux User Group) levelezőlistán üzenetet kaptam, amely végül ötletet adott nekem. Az üzenet, tulajdonképpen egy továbbított javaslat, a rendszermagmodul kiaknázását taglalta. Az emlegetett programmodul különösen visszataszító volt, mivel bizonyos rendszerhívások viselkedését módosította, hogy elrejtse magát az `lsmod` parancs elől, valamint láthatatlanná tegye a pásztázókat, jelszófeltörőket és szaglászó (sniffer), valamint a hasonló programokat. Kis híján felkiáltottam az irodámban: „Megtaláltam!” Nem kell rendszermagfoltozással bajlódnom, vagy újrafordítással és rendszerbetöltéssel bíbelődnöm; eszközkészítés gyanánt egyszerűen csak betölthető modult kell fejlesztenem. Felismertem, hogy a modulkihasználás mögött rejlő általános módszerbe a rendszerhívások mellé sokféle hasznos tevékenység beállítása belefér, ideértve a különböző biztonsági házirendeket, a finomabb hangolású biztonságtechnikát (mint amilyet a Unix lehetővé tesz), és természetesen a belső nyomkövető programomat.

Szia, rendszermag!

A rendszerhívások megváltoztatásának és kibővítésének érdekességeit cikkünk későbbi részében fogjuk megvizsgálni, elsőként azonban végezzünk bemelegítő ujjgyakorlatot egy rendszermag-példamodulon. Maga a mintaprogram igen egyszerű és a mindenki számára kedves első program rokona, ugyanakkor a betölthető rendszermodul két alapvető szolgáltatását képes szemléltetni: az `init_module` és a `cleanup_module` függvényeket:

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

int init_module() {
   printk("<1> Hello, rendszermag!\n");
   return 0;
}

void cleanup_module() {
   printk("<1> Nem sØrtıd m meg amiatt,"
          "hogy kit r ltØl a tÆrb l engem."
           A viszontlÆtÆsra!\n);
}
```

A fordításnál a `MODVERSION` elnevezésre a `#define`, ugyanakkor a ***linux/modversions.h*** állományra vonatkozóan az `#include` irányelvet kell használni a rendszer telepítésének megfelelően. Nevezzük parányi programunkat ***hello.c***-nek, és fordítsuk le a következő utasítással:

```
gcc -c -DMODULE -D __KERNEL__ hello.c
```

A fentiek után a jelenlegi könyvtárban létrejön a ***hello.o*** névvel rendelkező állomány. Amennyiben az X éppen működik, indítsuk el a parancsmódú konzolt, és rendszergazdaként gépeljük be az `insmod hello.o` parancsot. Ekkor a „Hello, rendszermag!" üzenetnek kell a képernyőn megjelennie. Annak ellenőrzésére, vajon a modul betöltődött-e a memóriába, használjuk az `lsmod` parancsot: ez megmutatja, hogy a modul betöltése megtörtént és a modul mekkora tárterületet foglal el. Most az `rmmod`-dal eltávolíthatjuk a szóban forgó modult: ekkor tájékoztatót kapunk arról, hogy a modul memóriából való törlése megtörtént. A ***linux/kernel.h*** és a ***linux/module.h*** függvényprototípus-állományok bármilyen programfejlesztés két legfontosabb állománya, így valószínű, hogy erre neked is szükséged lesz, bármilyen modult is fejlesztesz. A legelőnyösebb – a ***modversions.h***-tól eltérően –, ha ezek a függvényprototípus-állományok a ***/usr/include/linux*** könyvtárban találhatók, és nem a linuxos forráskód könyvtárfájában. A baj az, hogy a gyakorlat hajlamos nagyszabású leállásokhoz és fejfájáshoz vezetni. Bármilyen jelentősebb modulhoz számos további rendszermagprototípus-állományt kell használni, és a programmodulok fejlesztése közben a `grep -l /usr/include/linux` parancsot jó barátnak fogod tartani.

A hasznos „task_struct" elnevezései

Name	*Type*	*Description*
uid	uid_t	User ID
euid	uid_t	Effective user ID
gid	gid_t	Group ID
egid	gid_t	Effective group ID
pid	pid_t	Process ID
pgrp	pid_t	Process group ID
p_opptr	task_struct*	Original parent task
fs	fs_struct*	Filesystem information
blocked	sigset_t	Set of blocked signals

Gondolj a programodban az `init_module`-ra az elem létrehozójaként: ez a függvény lefoglalja a szükséges tárterületet, kezdőértéket ad a változóknak és úgy változtatja meg a rendszer állapotát, hogy a modulod is el tudja látni a feladatát. Ebben az `init_module` nem tesz egyebet, mint egyszerűen jelzi a jelenlétét, majd 0 (nulla) visszatérési értékkel mutatja működésének sikerességét, több C-függvényhez hasonlóan. Ezért a hello modul kezdeti értékadásában kizárólag a `printk` függvény hívása szerepel, amely különösen jól használható, rendelkezésünkre álló függvény. Lényegében a szabványos C-nyelvi függvénynek, a `printf`-nek megfelelően működik, két eltéréssel. Elsőként a `printk` a legszembetűnőbb módon lehetővé teszi, hogy az üzenet számára előnyben részesítési vagy rangsorolási számot adjunk meg: a relációs jelek közé zárt egyes számjegyet. Másodszor a `printk` a kimenetét folyamatosan a használt átmeneti területnek küldi, amelyet viszont a rendszermagnaplózó használ fel, s ezt ezen kívül a `syslogd` is megkapja. Minthogy a `syslog` kimenete gyakran ürítésre kerül, a gondosan kiválasztott helyre beszúrt, magas rangsorolási számú üzenettel a hibakeresést nagymértékben elősegíthetjük – különösen amiatt, hogy a rendszermag tárterületén levő hiba hajlamos a gép összeomlásához vagy a rendszermag kisebb rendellenességeihez vezetni.

Nos, miért is ne használnánk éppen a `printf`-et? A válasz egyszerű: nem használjuk, mert nem tudjuk használni. A Linux-rendszermag nem kapcsolódik a C-könyvtárhoz, emiatt az olyan jó öreg bútordarabok, amilyen a `printf` is, nem érhető el a rendszermag területén lévő kódban. Viszont magában a rendszermagban rengeteg könyvtári eljáráshoz hasonló utasítássorozat létezik, beleértve a C-programkönyvtár karakterláncot kezelő `str`-függvényeinek legtöbbjét.

A fent leírtaknak saját modulban történő használatához csak be kell fűzni a ***linux/string.h*** állományt – kérlek, vigyázz, nehogy a C-könyvtárbeli változatot fűzd be! Ha az `init_module`-t létrehozó modulnak tekintjük, akkor a `remove_module`-t eltávolító modulnak kell tekintenünk. Amennyire csak lehetséges, gondoskodj a program által hátrahagyott felesleges adatok eltávolításáról. Abban az esetben, ha nem tudsz tárterületet felszabadítani vagy adatszerkezetet helyreállítani, a normál üzemmódhoz való visszatérés végett a gépet újra kell indítanod.

Egy másik érdekes modul

Most rátérünk a haladó szintű nehézségekre. Az 1. lista (➲ http://www.linuxvilag.hu/Magazin.html/) olyan programmodult mutat be, amely naplóz minden `uid 0`-tól (rendszergazda) és `uid 500`-tól (jómagam a saját munkaállomásomon) eltérő felhasználótól érkező írási rendszerhívást, amennyiben valahol az ideiglenes tárterületen előfordul a Linux szó. Megeshet, egy kicsit erőlködni kell, hogy ennek a modulnak önmagában való hasznát belássuk, de kezeskedem, több hasznos elgondolást is bemutat. Mindezt úgy tehetjük meg, ha az írási rendszerhívást a saját függvényünkkel helyettesítjük, amely elvégzi az ellenőrzést, majd a naplózást és végül kiadja az írási kérelmet. Nos, rajta, haladjunk végig a példán lépésről lépésre!

Fordíts figyelmet valamennyi befűzendő (***include***) állományra. Természetesen seregnyi van belőlük, de azért mégsem kell kétségbe esnünk, bennünket most csak a ***linux/sched.h*** és az ***asm/uaccess.h*** állományok érdekelnek. A ***sched.h*** állomány lehetővé teszi a hozzáférést a pillanatnyi `task_struct` szerkezethez a `current` makróutasításon keresztül, valamint a pillanatnyi folyamat számos hasznos jellemzőjéhez (a `task_struct` által biztosított adatokat a ***táblázatunkban*** láthatod). Az ***uaccess.h*** állomány viszont a felhasználói adatterületről szolgáltat adatokat (részletes tájékoztatás alább található).

Még ez a néhány mező a `task_struct` adatszerkezetben is elegendő ahhoz, hogy néhány érdekes modult működőképessé tegyen. Legyen-e szabad tetszőleges felhasználónak a `su` paranccsal rendszergazdaként tevékenykednie? Ha szükséges, meggátolhatod ebben, becsomagolva a SETUID-ot, és ellenőrizve az előre beállított UID-okat, mielőtt a valódi azonosítók (SETUID) felbukkannak. Ennek segítségével a rendszermag szintjén alakíthat ki „kormánycsapatot" (`wheel group`), azaz olyan csoportot, amelynek tagjai a `su` parancson keresztül rendszergazdai jogokat gyakorolhatnak. A Szabad Program Alapítvány (Free Software Foundation – FSF) már régóta azon a véleményen van, hogy a `wheel group` a hatalomelvű rendszergazdák eszköze (bővebben a GNU `su` leírásában olvashatunk a témáról).

Az a képesség, hogy a rendszerhívások viselkedését csupán az azokat kezdeményező felhasználói azonosító (UID) alapján megvizsgáljuk vagy megváltoztassuk, nyilvánvalóan nagy teljesítmény. Megteremti a jó házirend alapjait, amelyben meg lehet szabni a „nobody" felhasználónak és társainak az uucp-, levelező- és Postgres adatbázis-felhasználók tevékenységi körét. Az említetteken kívül még hatékonyabb módszer is

létezik az átadott érték alapján a viselkedés megváltoztatására. Egyelőre figyelmen kívül hagyjuk a `sys_call_table` (rendszerhívások táblázata) és az `origwrite` rendszerváltozókat, és közvetlenül a `wrapped_write` függvényre térünk, amely nemcsak a hívó folyamat felhasználói azonosítóját (`uid`), de az átmeneti terület jellemzőjét is megvizsgálja. A legelső dolog, amelyet észre kell vennünk, az, hogy a `wrapped_write` pontosan egy `kmalloc` függvényhívással indul. Jó, de miért nem a `malloc`-ot használjuk? Gondoljunk vissza a fentebb mondottakra: még mindig a rendszermag területén vagyunk és nem férünk hozzá a `malloc` és egyéb szabványos könyvtárbeli függvényhez. De ha még hozzá is férnénk, minthogy a `malloc` hívása felhasználói területre utaló mutatót ad vissza, ez haszontalan lenne. A rendszermag területén kell helyet foglalni, ahová a `buf` értékből bemásolhatjuk az adatokat. Fontos ez a mozzanat: a tárban ugyanaz a rendszermag és a felhasználói adatterület közötti láthatósági korlát, amely megakadályozza a felhasználói programokat, hogy a rendszermag tartalmát módosítsák, valami hozzátesz valamennyit rendszermag-programozói gyakorlatunkhoz is. Amikor egy C-programból a `write` utasítást meghívjuk, egy felhasználói tárterületre utaló mutatót adunk át, amely a rendszermagból nem érhető el. Ezért amennyiben felhasználói adatterületen levő mutató által kijelölt adaton szeretnénk műveletet végezni, először is a rendszermag területére kell másolni. Ezt megteszi a `copy_from_user` makróutasítás, amely három értéket használ: a „hová” mutatót, a „honnan” mutatót és a számlálót.
A `wrapped_write` hátralevő része meglehetősen egyszerű, feltéve, hogy ismerjük a jelenlegi állapotot és a `task_struct` tartalmát. Talán egy ugyanilyen szórakoztató modul az `strstr` függvényt használná a „Pocsék Linux” karakterlánc keresésére, s ha megtalálta, a `write_buf` átváltoztatná. „A Linux a nyerő”-re, majd visszaírná a felhasználói adatterületre a `copy_to_user` makróutasítással, még az eredeti írási utasítás kiadása előtt. Ekkor, ha a gyanútlan felhasználó a „Pocsék Linux” kifejezést használta, most „A Linux a nyerő” karakterlánccal lesz helyettesítve. E helyen a `kfree` fontosságára kell felhívnom a figyelmet. A „szivárgó” memóriaterületek nemkívánatosak a rendszermagban, ezért fordítsunk nagy gondot arra, hogy `kfree`-vel mindent felszabadítsunk, amit a `kmalloc`-kal lefoglaltunk. Valójában az `init_module`-ban van elhelyezve az a kapcsoló, mely alapján a vezérlést az eredeti függvény helyett az általunk írott függvény kapja meg. Emlékezzünk vissza, hogy a `sys_call_table` tulajdonképpen a függvényekre utaló mutatók tömbje. A `SYS_write` indexértékének – vagyis az írás iránti rendszerhívásnak megfelelő állandó – megváltoztatásával az eredeti írási tevékenység újjal helyettesíthető. Minden esetben gondoskodj az eredeti függvény mentéséről, hogy a saját függvénytárból való törlésed után az eredetit vissza lehessen tölteni a tárba. Most bemutatott modulunkat ki lehet próbálni, ha fordítás után és telepítését elvégzed az `insmod`-dal, majd 0-tól és 500-tól eltérő felhasználói azonosítóval kiadod a `su` parancsot, és beírod a konzolon, hogy

```
%echo " n kedvelem a Linuxot."
```

Ekkor a rendszermag üzenetet fog küldeni, hogy már megint a Linuxot hoztad szóba. Gratulálunk!

Az utolsó példa

A 2. lista – amely egyébként az ➲ http://www.linuxvilag.hu/Magazin.html/ webcímen található meg – , egy olyan hasznos modult mutat be, amely segít megvédeni a gépünket, hogy ne essen veremromboló támadások áldozatául. Az ilyen támadások alapvetően rögzített hosszúságú ideiglenes tárterület végén túli területre kívánnak írni, ily módon felülírva a pillanatnyilag futó függvény visszatérési címét, amely rendszerint valamilyen végrehajtható állományra (***/bin/sh...***) való ugrás. Mivelhogy a `httpd`, `fingerd`, vagy `wu-ftpd` programoknál nincs értelme héjprogramokat futtatni, a következőkben ismertetünk egy módszert, amellyel ez a lehetőség kizárható. Mostanra már talán akkora gyakorlatra tettél szert, hogy a példában szereplő programkód java teljesen érthető lesz a számodra, mindössze talán egy kivétellel, a `strncpy_from_user` függvényt leszámítva. Ahogyan bizonyára már kitaláltad, ez a függvény szinte pontosan úgy működik, mint C-könyvtárbeli rokona, és okos eszköz arra, hogy a felhasználói tárterületről nullavégű karakterláncot hozzunk át. Mivel a programkód egyszerű, röviden át fogjuk tekinteni a megközelítést, és utána rád bízom, hogy újabb gondolataidat megvalósítva miképpen fokozod számítógépes rendszered biztonságát.
A 2. listában bemutatott programot könnyű üzembe állítani. Nem annyira hatékony vagy nagyteljesítményű, mint amekkorát talán elvárnánk. Mivel a programkód írásánál az egyértelműség volt a fő irányelv, a `wrapped_execve` modulban könnyű a soros keresést valami hatékonyabbra cserélni. Tulajdonképpen amit ez a modul tesz, nem más, minthogy újra meghatározza a `kill` rendszerhívásokat, téve ezt olyan módon, hogyha 42-es jelet küldök egy folyamatnak, akkor felkerül a „nem biztonságos” folyamatok listájára, vagyis egy olyan listára, amelyben a folyamatok nem engedik futni az `sh` betűket tartalmazó parancsokat. A 42 az egyik valós idejű jel, talán éppen használaton kívül van. Amennyiben mégis használod ezt a jelet, bátran helyettesítsd bármilyen 32 és 64 közötti számmal. Az `execve` ellenőrzi, vajon a folyamat biztonságos-e, és ha nem az, megpróbál-e héjprogramot indítani. Ha erre a kérdésre is igenlő válasz adható, akkor a program sikeresen befejeződik anélkül, hogy bármi történt volna. A kiszolgáló számítógép minden folyamatánál könnyű használni ezt a modult, csupán csak a `kill -42 ...` sort kell beszúrni az kezdeti értékeket beállító héjprogramokba.
A 2. lista fejlődésbeli lépést képvisel az 1. listához képest: megmutatja, hogy a rendszerhívási útvonalhoz nemcsak viselkedésmintát lehet adni, de a rendszerhívások lefutása is módosítható. Röviden: hasznos munkát végez nekünk. Remélem, a szerény példák mindenkit legalább annyira lázba hoztak, mint amennyire a rendszermag kiaknázásának lehetőségei engem felvillanyoztak, hogy növeljem a rendszer biztonságát. E cikk megadja az elinduláshoz szükséges segítséget, és a kapcsolódó címeken – részben Linux-programozók számára – a leírások bőséges tárháza található, amely segít az összetettebb és szolgáltatás-központúbb modulok megalkotásában.

William C. Benton
(willb@acm.org) egyetemi tanulmányait a University of Wisconsinon folytatta. Párhuzamos programok teljesítményfigyelése, valamint biztonsági nehézségek programnyelvi és fordítási megközelítése iránt érdeklődik.

Kapcsolódó címek

- ➲ http://www.linuxdoc.org/guides.html
- ➲ http://www.plex86.org
- ➲ http://user-mode-linux.sourceforge.net