Laborator 13 - Securitate

Nice to read

- TLPI - Chapter 9, Process Credentials
- TLPI - Chapter 36: Process Resource Usage

Securitate

(2.5 puncte) Utilizatori şi Grupuri

Fiecare utilizator are un nume de login unic şi un identificator numeric asociat - `UID`. Utilizatorii pot aparţine unuia sau mai multor grupuri. Fiecare grup are de asemenea un nume unic şi un identificator numeric de grup - `GID`.
Fişierele care definesc utilizatorii şi grupurile din sistem sunt următoarele:

/etc/passwd

- defineşte utilizatorii care se pot autentifica în sistem.
- detalii despre formatul fişierului `/etc/passwd`.

Doar utilizatorul privilegiat are dreptul să modifice fişierul `/etc/passwd`, restul utilizatorilor având doar drept de citire:

```
$ ls -al /etc/passwd
-rw-r--r-- 1 root root 1859 2011-04-21 22:15 /etc/passwd
```

Aplicaţie:
Listaţi toţi utilizatorii din sistem:

```
$ cat /etc/passwd
```

Listaţi intrarea asociată utilizatorului `student`:

```
$ grep student /etc/passwd
student:x:1000:1000:student,,,:/home/student:/bin/bash
```

Observaţi mai jos interpretarea fiecărui câmp:

Username	Password	UID	GID	Comment	Home dir	Shell
student	x	1000	1000	student,,,	/home/student/	/bin/bash/

Caracterul `x` din câmpul `Password` indică faptul că parola pentru utilizatorul `student` este ţinută în fişierul `/etc/shadow`.

/etc/shadow

- reţine parolele encriptate pentru utilizatorii din sistem.
- detalii despre formatul fişierului `/etc/shadow`.

Doar utilizatorul privilegiat are dreptul să citească fişierul `/etc/shadow`:

```
$ ls -al /etc/shadow
-rw-r----- 1 root shadow 1241 2011-04-21 22:15 /etc/shadow
```

Aplicaţie:
Listaţi intrarea asociată utilizatorului `student`:

```
$ sudo grep student  /etc/shadow
student:$bwS6SHeI$lG9l4RIb6Nq[....]:14738:0:99999:7:::
```

Observaţi mai jos interpretarea fiecărui câmp:

Username	Password	Last Changed	Minumum	Maximum	Warn	Inactive	Expire
student	6bwS6SHeI$lG9l4RIb6Nq[?.]	14738	0	99999	7		

/etc/group

- defineşte grupurile din sistem şi asocierea cu utilizatorii existenţi.
- detalii despre formatul fişierului `/etc/group`.

Aplicaţie:
Analizaţi grupurile existente în sistem:

```
$ less /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
[.....]
```

Listaţi grupurile din care utilizatorul `student` face parte:

```
$ groups student
student : student adm dialout cdrom plugdev lpadmin admin sambashare
```

Alternativ, inspectând fişierul `/etc/group`:

```
$ cat /etc/group | grep student
adm:x:4:student
dialout:x:20:student
cdrom:x:24:student
plugdev:x:46:stuent
lpadmin:x:105:student
admin:x:119:student
student:x:1000:
sambashare:x:122:student
```

Observaţi mai jos interpretarea fiecărui câmp:

Group Name	Password	GID	User List
dialout	x	20	student

(2.5 puncte) Privilegii. Setarea privilegiilor unui proces

Într-un sistem Linux fiecare proces are asociat un set de atribute care determină privilegiile acelui proces în timpul rulării. Printre atributele cele mai importante regăsim:

- `RUID`, `RGUID` - real `UID` şi real `GID` al userului din partea căruia un proces rulează.
- `EUID`, `EGUID` - effective `UID` şi effective `GID`, folosite pentru verificări de privilegii.
- `SUID`, `SGID` - saved `UID` şi saved `GID`, folosite pentru a suporta *schimbarea* de permisiuni.
- grupuri suplimentare, o listă de grupuri (GID) în care procesul este membru.
- `umask` - mască de biţi care determină atributele de control implicite la crearea unui obiect în sistemul de fişiere.
- `scheduling parameters` - fiecare proces are asociată o politică de planficare.
- `limits` - limitări ale resurselor per process.
- `filesystem root` - rădăcina sistemului de fişiere.

Aplicaţie:

- Analizaţi structura _task_struct din nucleul Linux, şi observaţi atributele asociate cu un proces.

Programe Set User ID

Ca orice alt fişier, un fişier executabil are asociat un utilizator şi un grup care definesc apartenenţa acelui fişier. În plus, un fişier executabil deţine 2 biţi de permisiune speciali:

- set user id bit - `setuid`.
- set group id bit - `setgid`.

Utilizatorii obişnuiţi care au drept de execuţie pe un program cu bitul `setuid` activat, pot rula acel program cu privilegiile deţinătorului programului.

Aplicaţie:
Observaţi ce se întâmplă cu EUID şi EGID pentru un program are are bitul `suid` dezactivat respectiv activat.

printid.c

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(void) {
    <a href="http://www.opengroup.org/onlinepubs/009695399/functions/printf.html">printf</a>(
        "Real      UID = %d\n"
        "Effective UID = %d\n"
        "Real      GID = %d\n"
        "Effective GID = %d\n",
        getuid (),
        geteuid(),
        getgid (),
        getegid()
    );
    return 0;
}
```

Rulare cu bitul `setuid` dezactivat:

```
student$ sudo su
[sudo] password for student:
root# ls
printid.c
root# gcc -Wall printid.c -o printid
root# exit
student$ ls -al printid
-rwxr-xr-x 1 root root 4894 2011-05-17 23:56 printid
student$ ./printid
Real      UID = 1000
Effective UID = 1000
Real      GID = 1000
Effective GID = 1000
```

Rulare cu bitul `suid` activat:

```
student$ sudo su
root# gcc -Wall printid.c -o printid
root# ls
printid  printid.c
root# chmod ug+s printid
root# exit
student$ ls -al printid
-rwsr-sr-x 1 root root 4894 2011-05-18 00:01 printid
student$ ./printid
Real      UID = 1000
Effective UID = 0
Real      GID = 1000
Effective GID = 0
```

În timpul rulării cu bitul `setuid` activat EUID respectiv EGID iau valoarile corespunzătoare UID şi GID ale utilizatorului care deţine fişierul executabil.

Aplicaţie:
Observaţi ce se întâmplă cu programul `ping` dacă bitul `setuid` este dezactivat:

```
student$ which ping
/bin/ping
student$ ls -al /bin/ping
-rwsr-xr-x 1 root root 34716 2010-07-28 11:44 /bin/ping
student$ sudo chmod u-s /bin/ping
student$ ls -al /bin/ping
-rwxr-xr-x 1 root root 34716 2010-07-28 11:44 /bin/ping
student$ ping google.ro
ping: icmp open socket: Operation not permitted
```

(2.5 puncte) Limitări ale resurselor

getrusage

Apelul de sistem getrusage ne permite să aflăm statistici despre procesul curent sau despre copiii/threadurile acestuia.

```
int getrusage(int who, struct rusage *res_usage);
```

Prin parametrul `who` se specifică despre cine se vor obţine informaţiile:

- RUSAGE_SELF - procesul curent
- RUSAGE_CHILDREN - toate procesele copil care s-au încheiat şi care au fost aşteptate (wait)
- RUSAGE_THREAD - threadul apelant (opţiune valabilă din Linux 2.6.26)

Resursele sunt întoarse printr-un pointer la o structură `struct rusage`:

```
struct rusage {
               struct timeval ru_utime; /* user time used */
               struct timeval ru_stime; /* system time used */
               long   ru_maxrss;        /* maximum resident set size */
               long   ru_ixrss;         /* integral shared memory size */
               long   ru_idrss;         /* integral unshared data size */
               long   ru_isrss;         /* integral unshared stack size */
               long   ru_minflt;        /* page reclaims */
               long   ru_majflt;        /* page faults */
               long   ru_nswap;         /* swaps */
               long   ru_inblock;       /* block input operations */
               long   ru_oublock;       /* block output operations */
               long   ru_msgsnd;        /* messages sent */
               long   ru_msgrcv;        /* messages received */
               long   ru_nsignals;      /* signals received */
               long   ru_nvcsw;         /* voluntary context switches */
               long   ru_nivcsw;        /* involuntary context switches */
           };
```

Informaţiile despre un proces copil se pot afla şi cu funcţiile wait3 şi wait4.

Aplicaţie:

- Folosiţi utilitarul time (varianta oferită de GNU ? urmăriţi secţiunea **GNU VERSION**) pentru a afla numărul de *page fault*-uri şi numărul de *context switch*-uri (voluntare şi involuntare) pentru următoarele comenzi (şirul `-format afisare-` trebuie înlocuit cu un format valid de afişare pentru *page fault*-uri şi *context switch*-uri):
  - /usr/bin/time --format " -format afisare- " wget elf.cs.pub.ro/so/wiki
  - /usr/bin/time --format " -format afisare- " // TODO //

- Folosiţi strace pentru a afla **ce apel de sistem** foloseşte utilitarul time cu scopul de a obţine aceste informaţii.
  - **Hint**: Căutaţi şirul `ru_utime`. Consultaţi pagina de manual a apelului de sistem.

getrlimit/setrlimit

Orice proces are un set de limite pentru resursele pe care le poate folosi. În acest fel ne putem asigura spre exemplu că un anumit program nu consumă excesiv de multe resurse (fapt ce ar putea provoca îngreunarea sistemului sau chiar blocarea acestuia).

Apelurile de sistem prin care putem afla sau modifica aceste limite sunt getrlimit şi setrlimit:

```
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

Argumentul `resource` identifică resursa întoarsă sau schimbată. Argumentul `rlimit` reprezintă un pointer la o structură de tip `struct rlimit` care arată în felul următor:

```
struct rlimit {
    rlim_t rlim_cur;  /* Soft limit (actual process limit) */
    rlim_t rlim_max;  /* Hard limit (ceiling for rlim_cur) */
};
```

Limita *soft* reprezintă cât poate să consume acel proces din resursa respectivă. Limită soft poate fi setată de proces oriunde de 0 la `rlimit_max`. Limita *hard* reprezintă valoarea maximă pe care o poate atinge limita *soft*. Un proces privilegiat (CAP_SYS_RESOURCE) poate modifica limita hard, atât timp cât 0 < lim_soft < lim_hard, dar un proces neprivilegiat nu poate schimba decât limita soft.

Aceste limite sunt moştenite de procese în urma apelului `fork` şi se păstrează şi în urma apelurilor de tip `execv`.

Aceste valori sunt citite/scrise în fişierul `/proc/PID/limits`.

Aplicaţie

- Aflaţi limitele soft şi hard are proceslui shell în care vă aflaţi:
  - sudo cat /proc/$$/limits

ulimit

Utilitarul ulimit are rolul de a scrie/citi valorile limitelor pentru procesul shell.

Aplicaţie:

- Folosiţi ulimit pentru a preveni un fork bomb:
  - Limitaţi numărul de procese din shell-ul vostru ulimit -u 300
  - Verificaţi că aceste valori s-au setat verificând fişierul `/proc/$$/limits`
  - Rulaţi, **în aceeaşi consolă**, următorul program care creează un fork-bomb: :(){ :|:& };:
  - Mai puteţi crea procese în acel terminal?
  - Afişaţi din altă consolă procesele bash create

Explicatie fork bomb

(2.5 puncte) Izolarea unui process

chroot

chroot reprezintă un mecanism prin schimbarea directorului rădăcină al unui proces. Un proces al cărui director chroot este `/var/lib/my/` nu va putea accesa intrări din sistemul de fişiere aflate în afara acestei ierarhii. Operaţiunea de schimbare a directorului rădăcină folosind `chroot` poartă şi numele de *jailing*.

Pentru a urmări directorul rădăcină al unui proces se recomandă folosirea lsof. Directorul rădăcină este identificat prin numele `rtd` (*root directory*). În marea parte a proceselor, acesta este chiar rădăcina sistemului de fişiere (`/`) ? adică un proces nu este *chrooted*.

Se poate schimba directorul rădăcină folosind comanda chroot.

Aplicaţii:

1. Folosiţi `lsof -p $$` pentru a urmări directorul rădăcină al procesului shell curent.
2. (lucraţi ca **root**) Instalaţi `postfix` (`apt-get install postfix`).
   - Folosiţi comanda `pstree -p -h $(pgrep master)` pentru a lista subarborele aferent proceselor Postfix.
   - Folosiţi `lsof` pentru a urmări directorul rădăcină pentru procesele aferente. Ce observaţi?

Din punctul de vedere al procesului, directorul său rădăcină este `/` şi toate resursele accesate trebuie să se regăsească în acea ierarhie. Astfel, dacă un proces are nevoie de biblioteci sau fişiere de configurare, acestea trebuie să se regăsească în noua ierarhie.

Aplicaţii:

- Intraţi în directorul `chroot/` din arhiva de resurse a laboratorului.

1. Folosiţi make pentru a compila. Veţi obţine executabilele `hello_static` şi `hello_dynamic`.
   - Ca **root**, rulaţi comenzile:chroot . ./hello_static respectiv chroot . ./hello_dynamic
     - **Atenţie!** Există un caracter `.` (punct, dot) între `chroot` şi `./hello_static` (respectiv `./hello_dynamic`), semnificând directorul curent.
     - Ce observaţi?
2. Rulaţi, chroot-at în directorul curent, un proces care porneşte din executabilul `/bin/bash`.
   - Creaţi, în directorul local, directoarele locale necesare (`bin/`, `lib/`, `lib64/`).
   - Copiaţi executabilul aferent `bash` în `bin/`.
   - Copiaţi bibliotecile necesare.
     - **Hint**: folosiţi `ldd`.
   - Copiaţi executabilul aferent comenzii `ls` în `bin/`.
   - Ca **root**, rulaţi comandachroot .

Funcţionalităţile specifice `chroot` sunt implementate cu ajutorul unui apel de sistem, denumit tot chroot. Acest apel schimbă directorul rădăcină al unui proces. Procesul care apelează `chroot` trebuie să fie privilegiat (trebuie să fie deţinut de `root` sau să aibă capabilitatea `CAP_SYS_CHROOT` activată).

```
int chroot(const char *path);
```

Aplicaţii:

- Intraţi în directorul `chroot/` din arhiva de resurse a laboratorului.

1. Folosiţi make pentru a compila. Veţi obţine executabilul `do_chroot`.
   - Urmăriţi fişierul sursă `do_chroot.c`.
   - Creaţi directorul identificat de macro-ul `CHROOT_FOLDER` în fişierul sursă `do_chroot.c`.
   - Ca **root**, rulaţi executabilul obţinut. Ce observaţi?
   - Creaţi, în directorul `CHROOT_FOOLDER` fişierul identificat de macro-ul `CHROOTED_FILE`.
     - Ca **root**, rulaţi, din nou, executabilul obţinut. Ce observaţi?
2. Porniţi de la exemplul anterior pentru a crea un program care răspundă la întrebarea: "Este accesibil, prin intermediul unui file descriptor, un fişier deschis înainte de apelul chroot(2) aflat în afara jail-ului?".

Resurse utile

- Wikipedia about setuid
- Wikipedia about UID
- Securizarea şi monitorizarea sistemului - USO