

Laborator 13 - Securitate

Nice to read

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

Securitate

Utilizatori și Grupuri

Fiecare utilizator are un nume de login unic și un identificator numeric asociat - **UID**. Utilizatorii pot aparține unuia sau mai multor grupuri. Fiecare grup are de asemenea un nume unic și un identificator numeric de grup - **GID**.

Fișierele care definesc utilizatorii și grupurile din sistem sunt următoarele:

/etc/passwd

- definește utilizatorii care se poate loga în sistem.
- understanding `/etc/passwd` [file format](#).

Doar utilizatorul privilegiat poate să modifice fișierul `/etc/passwd`, restul având doar drept de citire:

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

Aplicație:

Listați toți utilizatorii din sistem;

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

Listați intrarea asociată utilizatorului `student`:

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

Observați 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 ținută în fișierul `/etc/shadow`.

/etc/shadow

- reține parolele encriptate pentru utilizatorii din sistem.
- understanding `/etc/shadow` [file format](#).

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

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

Aplicație

Listați intrarea asociată utilizatorului `student`:

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

Observați mai jos interpretarea fiecărui câmp:

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

/etc/group

- definește grupurile din sistem și asocierea cu utilizatorii existenți.
- understanding /etc/group [file](#).

Aplicație:

Analizați grupurile existente în sistem:

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

Listați grupurile din care utilizatorul `student` face parte:

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

Alternativ, inspectând fișierul `/etc/group`:

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

Observați mai jos interpretarea fiecărui câmp:

Group Name	Password	GID	User List
dialout	x	20	student

Privilegiile. Setarea privilegiilor unui proces

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

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

Aplicație

- Analizați structura [task_struct](#) din nucleul Linux, și observați atributele asociate cu un proces.

Programe Set User ID

Ca orice alt fișier, un fișier executabil are asociat un utilizator și un grup care definesc apartenența aceluia fișier. În plus, un fișier executabil deține 2 biți de permisiune speciali:

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

Utilizatorii obișnuiți care au drept de execuție pe un program cu bitul `setuid` activat, pot rula acel program cu privilegiile deținătorului programului.

Aplicație:

Observați ce se întâmplă cu EUID și EGID pentru un program care 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 valorile corespunzătoare UID și GID utilizatorului care deține fișierul executabil.

Aplicație Observați 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
```

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 obține informațiile:

- RUSAGE_SELF - procesul curent
- RUSAGE_CHILDREN - toate procesele copil care s-au încheiat și care au fost așteptate (wait)
- RUSAGE_THREAD - threadul apelant (opțiune 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 */
};
```

Informațiile despre un proces copil se pot afla și cu funcțiile [wait3](#) și [wait4](#).

Aplicație:

- Folosiți utilitarul [time](#) (variantea oferită de GDB) pentru a afla numărul de page-fault-uri și numărul de context-switch-uri (voluntare și involuntare) pentru următoarele comenzi:
 - `/usr/bin/time --format "%format afisare%" wget elf.cs.pub.ro/so/wiki`
 - `/usr/bin/time --format "%format afisare%" // TODO //`
- Folosiți [strace](#) pentru a afla **ce apel de sistem** folosește utilitarul [time](#) cu scopul de a obține aceste informații.

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:

```
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 `rlim_max`. Limita *hard* reprezintă valoarea maximă pe care o poate atinge limita *soft*. Un proces privilegiat (`CAP_SYS_RESOURCE`) poate modifica limita *hard*, atâta timp cât ($0 < \text{lim_soft} < \text{lim_hard}$), dar un proces neprivilegiat nu poate schimba decât limita *soft*.

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

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

Aplicație

- Aflați limitele soft și hard ale procesului shell în care vă aflați:
 - `sudo cat /proc/$$/limits`

ulimit

Utilitarul [ulimit](#) are rolul de a scrie/citi valorile limitelor pentru procesul shell.

Aplicație:

- Folosiți `ulimit` pentru a preveni un fork bomb:
 - Limitați numărul de procese din shell-ul vostru `ulimit -u 300`
 - Verificați că aceste valori s-au setat verificând fișierul `/proc/$$/limits`
 - Rulați următorul program care creează un fork-bomb: `:{ } :|& };`
 - Mai puteți crea procese în acel terminal?
 - Afișați din altă consolă procesele bash create

Explicatie fork bomb

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 fișiere aflate în afara acestei ierarhii. Operațiunea de schimbare a directorului rădăcină folosind `chroot` poartă numele și de *jailing*.

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

Aplicații:

1. Folosiți `lsof -p $$` pentru a urmări directorul rădăcină al procesului shell curent.
2. (lucrați ca **root**) Instalați `postfix` (`apt-get install postfix`).
 - Folosiți comanda `pstree -p -h $(pgrep master)` pentru a lista subarborile aferent proceselor Postfix.
 - Folosiți `lsof` pentru a urmări directorul rădăcină pentru procesele aferente. Ce observați?

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

Aplicații:

- Intrați în directorul `chroot/` din [arhiva de resurse a laboratorului](#).
1. Folosiți `make` pentru a compila. Veți obține executabilele `hello_static` și `hello_dynamic`.
 - Ca **root**, rulați comenzile `chroot . /hello_static`, respectiv `chroot . /hello_dynamic`.
 - Ce observați?
 2. Rulați `/bin/bash chroot-at`.

- Creați, în directorul local, directoarele locale necesare (bin/, lib/, lib64/).
- Copiați executabilul aferent bash în bin/.
- Copiați bibliotecile necesare.
 - **Hint:** folosiți ldd.
- Copiați executabilul aferent comenzii ls în bin/.
- Ca **root**, rulați comandachroot . Toy around 😊

Funcționalitățile 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 deținut de root sau să aibă capabilitatea CAP_SYS_CHROOT activată).

Aplicații:

- Intrați în directorul chroot/ din [arhiva de resurse a laboratorului](#).
1. Folosiți make pentru a compila. Veți obține executabilul do_chroot.
 - Urmăriți fișierul sursă do_chroot.c.
 - Creați directorul identificat de macro-ul CHROOT_FOLDER în fișierul sursă do_chroot.c.
 - Ca **root**, rulați executabilul obținut. Ce observați?
 - Creați, în directorul CHROOT_FOLDER fișierul identificat de macro-ul CHROOTED_FILE.
 - Ca **root**, rulați, din nou, executabilul obținut. Ce observați?
 2. Porniți de la exemplul anterior pentru a crea un program care răspundă la întrebarea: "Este accesibil, prin intermediul unui file descriptor, un fișier deschis înainte de apelul chroot(2) aflat în afara jail-ului?".

Pentru detalii legate de *chroot escaping* recomandăm parcurgerea [discuției de aici](#).

Resurse utile

- [Wikipedia about setuid](#)
- [Wikipedia about UID](#)
- [Securizarea și monitorizarea sistemului - USO](#)

From:
<http://elf.cs.pub.ro/so/wiki/> - **Sisteme de Operare**

Permanent link:
<http://elf.cs.pub.ro/so/wiki/laboratoare/laborator-13>

Last update: 2011/05/20 09:16