

Securitate Software

V. Vulnerabilități specifice sistemelor de operare

Execuție cod cu prea multe privilegii

Objective

- prezentarea vulnerabilităților ce rezultă din manipularea greșită a permisiunilor aplicațiilor
- prezentarea mecanismelor de asignare a privilegiilor (UNIX) și vulnerabilitățile asociate

- 1 Vulnerabilitatea "prea multe privilegii"
- 2 Linux privilegii si vulnerabilitati
 - Sistemul de privilegii
 - Procese in linux
 - Programe privilegiate
 - Funcții pentru User și Group ID
 - Utilizarea gresita a privilegiilor
 - Scaderea permanenta a privilegiilor
 - Scaderea temporara a privilegiilor
 - Audit pentru cod ce modifica drepturile
 - Extensia privilegiilor

Test

1. Analizați următoarea secvență de cod. Corectați eventualele greșeli. Argumentați. (trimiteti raspunsul la *mihai.suciu [at] ubbcluj.ro*)

```
int get_user(char *user) {  
    char buf[1024];  
  
    if (strlen(user) > sizeof(buf))  
        die("error: _user_string_too_long\n");  
  
    strcpy(buf, user);  
}
```

Analizați următoarele secvențe de cod. Corectați eventualele greșeli. Argumentați.

(trimiteți răspunsul la *mihai.suciu [at] ubbcluj.ro*)

- din perspectiva cursului 2

```
int get_user(char *user) {
    char buf[1024];

    if (strlen(user) > sizeof(buf))
        die("error:_user_string_too_long\n");

    strcpy(buf, user);
}
```

- din perspectiva cursului 3

```
...
while (count) {
    if (strlen(nameStr) + count < (MAX_LEN - 1)) {
        ...
        strncat(nameStr, (char*)indx, count);
        ...
    }
}
nameStr[strlen(nameStr)-1] = 0;
```

Vulnerabilitatea "prea multe privilegii" - descriere

- o aplicație ce rulează cu privilegii mai mari decât are nevoie
 - cazul extrem: privilegii de administrator / system
- defect de design
 - contrazice principiul "cel mai mic privilegiu"
 - o aplicație primește nivelul minim de privilegii necesar pentru a-și face treaba
 - contrazice principiul "apărare în profunzime"
- poate fi un defect de implementare
 - când aplicația nu își coboară nivelul de privilegii atunci când ar trebui să o facă
- efecte:
 - dă atacatorului mai multă putere când este exploatată aplicația
 - ex. atacatorul poate executa cod cu privilegiile aplicației exploatate
 - ex. atacatorul poate accesa date pentru care în mod normal nu are privilegii

Referințe CWE

- CWE-264: "Permissions, Privileges, and Access Controls"
 - foarte generală
 - legată de administrarea permisiunilor, privilegiilor și altor caracteristici ce permit controlul accesului
- CWE-265: "Privilege / Sandbox Issues"
 - asignarea, administrarea, manipularea greșită a privilegiilor
- CWE-250: "Execution with Unnecessary Privileges"
 - execută operații la un nivel de privilegii mai mare decât este necesar
 - se creează o vulnerabilitate sau se amplifică efectul vulnerabilităților existente
- CWE-269: "Improper Privilege Management"
 - nu se creează, modifică, urmaresc, verifică corect drepturile de acces pentru un utilizator, se creează neintenționat o sfera de acces pentru acest utilizator
- CWE-271: "Privilege Dropping / Lowering Errors"
 - nu se modifică drepturile de acces (nu se scad privilegiile) pentru utilizatorii / resursele ce nu au privilegiile aplicației

Vulnerabilități

- folosirea de reguli stricte pe anumite resurse
 - ex. acces root / administrator doar pentru anumite fișiere sau directoare
 - \implies forțează ca aplicațiile să ruleze cu privilegii elevate pentru a putea avea acces la aceste resurse
 - nu permite utilizatorilor cu drepturi de acces limitate (și aplicațiilor acestora) să acceseze aceste resurse
- nu se coboară privilegiile înainte de a executa acțiuni pentru utilizatorii cu privilegii inferioare
 - uită să se facă asta
 - nu se apelează funcțiile care trebuie sau nu se specifică corect parametrii
 - nu se verifică dacă execuția funcțiilor apelate s-a terminat cu succes

Identificarea vulnerabilității

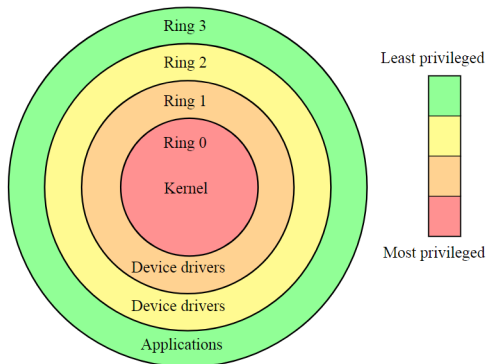
- în general
 - determinați dacă aplicația poate rula corect fără drepturi de administrator (*non-admin privileges*)
- code review
 - determinați ce privilegii necesită aplicația
 - determinați dacă drepturile de acces sunt configurate corect
- tehnici de testare
 - anulați privilegiile aplicației
 - ex. în Windows, luați jetonul aplicației (token) și parsați-l, instrumente precum Process Explorer (access token = obiect ce descrie contextul de securitate al unui proces / thread)

Recomandări

- rulați aplicația cu cel mai mic privilegiu
- determinați și înțelegeți de ce privilegii are nevoie aplicația
- scoateți privilegiile inutile
- poate fi un proces complex
 - mai ales atunci când aplicația interacționează cu alte aplicații ce au privilegii înalte

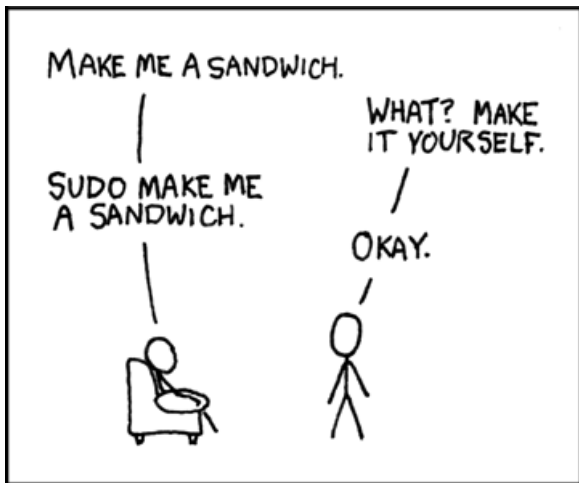
Privilegii - recapitulare

- inele de protecție



- utilizator \rightarrow *user ID* (UID)
- $UID = 0 \rightarrow$ "superutilizator" sau utilizator *root*
- grupuri identificate prin *group ID* (GID)
- utilizatorii */etc/passwd*, grupurile */etc/group*

Privilegii sudo (root)



Linux - privilegii și vulnerabilități, proces

- program
- proces
- în mod normal fiecare proces rulează cu privilegiile utilizatorului de care aparține
 - *user identity (UID)* este asociat cu toate procesele utilizatorului \Rightarrow *real UID (RUID)* pentru fiecare proces
- procesele ce aparțin *root (UID=0)* au acces absolut la toate resursele
- există cazuri în care un proces rulează cu alte privilegii (mai mari) decât ale utilizatorului de care aparține
 - procesele rulează ca și cum ar aparține de alt utilizator
 - *effective UID* determină privilegiile acestui proces
- privilegii speciale pot fi obținute prin mecanisme precum
 - set-user-id (SUID) și set-group-id (SETGID)

UID asociat unui proces

- identificatorul pentru utilizatorul unui proces (UID)
 - real UID
 - saved SUID
 - effective UID
- atunci când un proces rulează un program nou, ex. prin apelul `system execve()`
 - real UID rămâne la fel
 - effective UID se schimbă dacă
 - SUID este setat pentru noul program \implies effective UID primește UID al 'proprietarului' acelui program
 - SUID salvat este înlocuit de noul effective UID
- în mod normal un proces poate să-și schimbe *effective UID*
 - cu *real UID* sau *saved SUID*
- procesele cu *effective UID* 0 au acces complet la sistem

GID asociate unui proces

- identificatorii grupului unui proces (GID - group identifiers)
 - real GID
 - saved SGID
 - effective GID
 - supplemental GID, alte grupuri de care aparține utilizatorul curent
 - procesele cu *effective GID* 0 nu au acces complet asupra sistemului

Privilegiile unui proces

- permisiunile unui proces sunt determinate de:
 - *effective UID* al acelui proces
 - *effective GID* al acelui proces
 - *supplemental GID* al acelui proces

Programe SUID și SGID

- proces *effective UID / SGID = UID / GID* deținătorul programului
- *SUID / SGID* salvat pentru un proces = *effective UID / GID*
- permite utilizatorilor de rând (*normal user*) să acceseze resursele altor utilizatori
- proces
 - **pornește cu privilegii elevate**
 - poate să-și **coboare temporar privilegiile** prin trecerea la *UID / GID* său
 - poate să-și **recapete privilegiile** prin trecerea la *SUID / SGID*
 - poate să-și **coboare permanent privilegiile** prin înlocuirea *ID*-urilor efective și *ID*-urilor salvate cu *ID*-ul său real
- programe *SUID / SGID* non-root, ex *wall*
 - acțiuni limitate la operațiile descrise mai sus
- programe *SUID / SGID* root, ex. *ping*, *passwd*
 - majoritatea programelor *SUID* sunt *SUID root*, aparțin root
 - pot să-și schimbe *ID*-ul arbitrar

Exemplu de programe SUID și SGID non-root

```
$ find /usr/bin -perm /06000 -print0 | xargs -0 ls -l
-rwsr-sr-x 1 daemon daemon 51464 Jan 15 2016 /usr/bin/at
-rwxr-sr-x 1 root tty 14752 Mar 1 2016 /usr/bin/bsd-write
-rwxr-sr-x 1 root shadow 62336 May 17 02:37 /usr/bin/chage
-rwsr-xr-x 1 root root 49584 May 17 02:37 /usr/bin/chfn
-rwsr-xr-x 1 root root 40432 May 17 02:37 /usr/bin/chsh
-rwxr-sr-x 1 root crontab 36080 Apr 6 2016 /usr/bin/crontab
-rwxr-sr-x 1 root shadow 22768 May 17 02:37 /usr/bin/expiry
-rwsr-xr-x 1 root root 75304 May 17 02:37 /usr/bin/gpasswd
-rwxr-sr-x 1 root mlocate 39520 Nov 18 2014 /usr/bin/mlocate
-rwsr-xr-x 1 root root 32944 May 17 02:37 /usr/bin/newgidmap
-rwsr-xr-x 1 root root 39904 May 17 02:37 /usr/bin/newgrp
-rwsr-xr-x 1 root root 32944 May 17 02:37 /usr/bin/newuidmap
-rwsr-xr-x 1 root root 54256 May 17 02:37 /usr/bin/passwd
-rwsr-xr-x 1 root root 23376 Jan 18 2016 /usr/bin/pkexec
.....
```

Daemons

- procese ce rulează în fundal și furnizează servicii sistem
- pornite automat (*boot time*)
- adesea rulează ca și *root* pentru a permite efectuarea anumitor operații
- adesea ele rulează alte programe pentru a gestiona anumite sarcini, copiii lor sunt porniți cu privilegii *root*
- pot primi temporar identitatea altor utilizatori pentru anumite acțiuni într-o manieră sigură
 - atât timp cât programul își lasă *SUID* și *UID* la valoarea 0, își poate recăpăta privilegiile *root*
- exemplu: */bin/login*

Exemple de procese Daemon

```
$ ps axlf | grep -i tty
F  UID      PID  PPID  PRI  NI   VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
0  1000    1435   1386   20   0  14224    944 pipe_w  S+   pts/0
4      0    1105      1   20   0  65832   3464 -        Ss   tty1    0:00 /bin/login --
4  1000    1333   1105   20   0  22604   5200 wait_w  S+   tty1    0:00 \_ -bash
```

TIME COMMAND

0:00

_ grep -- color=auto -i tty

0:00 /bin/login --

0:00 _ -bash

Funcția seteuid()

- sintaxa

```
int seteuid(uid_t euid);
```

- schimbă *effective UID* pentru proces
- folosită pentru a schimba temporar privilegiile
- un proces ce are *effective UID*
 - *root*: poate să-și modifice *effective UID* la orice valoare
 - *non-root*: poate doar să-și schimbe *effective UID* între *saved SUID* și *real UID*

Funcția seteuid() - exemple

- utilizator cu $UID = 1000$ rulează un program *SUID* al utilizatorului $UID = 2$

```
$ ls -l suid-program  
-rwsrwsr-x 1 bin bin 8936 oct 29 12:50 suid-program
```

```
$ ./suid-program  
INITIAL
```

```
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
seteuid(33); // nu se poate schimba effective uid  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
seteuid(1000);  
real_uid = 1000, effective_uid=1000, saved_suid=2
```

```
seteuid(2);  
real_uid = 1000, effective_uid=2, saved_suid=2
```

Funcția seteuid() - exemple (II)

- utilizator cu *UID* = 1000 rulează un program *SUID* al utilizatorului root *UID* = 0

```
$ ls -l suid-program  
-rwsrwsr-x 1 root root 8936 oct 29 12:50 suid-program
```

```
$ ./suid-program  
INITIAL  
real_uid = 1000, effective_uid=0, saved_suid=0
```

```
seteuid(33); // se poate schimba effective UID  
real_uid = 1000, effective_uid=33, saved_suid=0
```

```
seteuid(1000);  
real_uid = 1000, effective_uid=1000, saved_suid=0
```

```
seteuid(33); // nu se poate schimba effective UID  
real_uid = 1000, effective_uid=1000, saved_suid=0
```

```
seteuid(0);  
real_uid = 1000, effective_uid=0, saved_suid=0
```

Funcția `setuid()` - descriere

- sintaxa

```
int setuid(uid_t uid);
```

- pentru **proces root**

- **schimbă toate UID**, *effective UID*, *real UID*, *saved UID* la *UID* specificat
- folosit pentru a primi permanent rolul unui utilizator
 - **scaderea drepturilor**

- pentru **proces non-root**: diferit în funcție de varianta Linux

- Linux, Solaris și OpenBSD: se comportă ca *seteuid()*
- FreeBSD și NetBSD: similar cu procesele root
- nu se recomandă a fi folosită pentru a diminua permanent privilegiile unui proces non-root

Funcția `setuid()` - exemple

- utilizator cu $UID = 1000$ rulează un program *SUID* al utilizatorului $UID = 2$

```
$ ls -l suid-program  
-rwsrwsr-x 1 bin bin 8936 oct 29 12:50 suid-program
```

```
$ ./suid-program  
INITIAL  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setuid(33); // nu se poate schimba  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setuid(1000);  
real_uid = 1000, effective_uid=1000, saved_suid=2
```

```
setuid(2); // se schimba doar effective UID  
real_uid = 1000, effective_uid=2, saved_suid=2
```

Funcția `setuid()` - exemple (II)

- utilizator cu $UID = 1000$ rulează un program *SUID* al utilizatorului $root$ $UID = 0$

```
$ ls -l suid-program  
-rwsrwsr-x 1 root root 8936 oct 29 12:50 suid-program
```

```
$ ./suid-program  
INITIAL  
real_uid = 1000, effective_uid=0, saved_suid=0
```

```
setuid(33); // scade permanent privilegiile  
real_uid = 33, effective_uid=33, saved_suid=33
```

```
setuid(1000); // nu se poate  
real_uid = 33, effective_uid=33, saved_suid=33
```

```
setuid(0); // nu se poate  
real_uid = 33, effective_uid=33, saved_suid=33
```

Funcția `setresuid()` - descriere

- sintaxa

```
int setresuid(uid_t ruid, uid_t euid, uid_t suid);
```

- folosită pentru a seta explicit cele trei UID
- dacă se dă ca și argument "-1", valoarea curentă corespunzătoare UID-urilor este salvată
- procesele root pot seta UID la orice valori
- procesele non-root pot configura orice ID la valoarea curentă a oricărui din cele trei UID

Funcția `setresuid()` - exemple

- utilizatorului cu *UID* = 1000 rulează un program *SUID* al utilizatorului cu *UID* = 2 și schimbă permanent *UID* = 1000

```
$ ls -l suid-program  
-rwsrwsr-x 1 bin bin 8936 oct 29 12:50 suid-program
```

```
$ ./suid-program  
INITIAL
```

```
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setresuid(-1, -1, -1);  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setresuid(33, 33, 33); // nu se poate  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setresuid(-1, 1000, -1);  
real_uid = 1000, effective_uid=1000, saved_suid=2
```

```
setresuid(-1, 2, -1);  
real_uid = 1000, effective_uid=2, saved_suid=2
```

Funcția setresuid() - exemple (II)

```
setresuid(-1, -1, 1000);  
real_uid = 1000, effective_uid=2, saved_suid=1000
```

```
setresuid(-1, -1, 2);  
real_uid = 1000, effective_uid=2, saved_suid=2
```

```
setresuid(-1, 1000, 1000); // schimba la UID = 1000  
real_uid = 1000, effective_uid=1000, saved_suid=1000
```

```
setresuid(-1, 2, 2); // nu se poate schimba inapoi la UID = 2  
real_uid = 1000, effective_uid=1000, saved_suid=1000
```

Funcția `setresuid()` - exemple (III)

- utilizatorul cu *UID* = 1000 rulează un program *SUID* al utilizatorului root *UID* = 2 și își schimbă permanent *UID* = 2

```
$ ls -l suid-program
-rwsrwsr-x 1 bin bin 8936 oct 29 12:50 suid-program
$ ./suid-program
INITIAL
real_uid = 1000, effective_uid=2, saved_suid=2
setresuid(-1, -1, -1);
real_uid = 1000, effective_uid=2, saved_suid=2
setresuid(33, 33, 33); // nu se poate
real_uid = 1000, effective_uid=2, saved_suid=2
setresuid(-1, 1000, -1);
real_uid = 1000, effective_uid=1000, saved_suid=2
setresuid(-1, 2, -1);
real_uid = 1000, effective_uid=2, saved_suid=2
setresuid(-1, -1, 1000);
real_uid = 1000, effective_uid=2, saved_suid=1000
setresuid(-1, -1, 2);
real_uid = 1000, effective_uid=2, saved_suid=2
```

Funcția setresuid() - exemple (IV)

```
setresuid(2, -1, -1); // schimba la UID = 2  
real_uid = 2, effective_uid=2, saved_suid=2  
setresuid(1000, -1, -1); // nu poate schimba la UID = 1000  
real_uid = 2, effective_uid=2, saved_suid=2
```

Funcția *setreuid()*

- sintaxa

```
int setreuid( uid_t ruid , uid_t euid );
```

- folosită pentru a seta *UID* real și efectiv
- dacă se dă ca și argument valoarea "-1" se folosește valoarea curentă pentru *UID*
- procesele root: pot configura ID la orice valoare
- procesele non-root: depinde de sistemul de operare, dar în general
 - *real UID* – > *effective UID*
 - *effective UID* – > *real UID*, *effective UID* sau *saved UID*
 - *saved UID* se încearcă sa fie actualizat daca *effective UID* diferit de noul *real UID*
- folosite în următoarele situații
 - un program nou are două *UID* deoarece *real UID* și *saved SUID* nu sunt root
 - programul vrea să scadă un set de privilegii
 - funcția *setresuid()* nu este disponibilă
 - soluție: *setreuid(getuid(), getuid());*

Funcții Group ID

- *setegid*: schimb între *effective GID* și *saved SGID* sau *real GID*
- *setgid*: schimbă *effective GID*, posibil și *saved SGID* și *real GID*
- *setresgid*: schimbă toate *GID*
- *setregid*: schimbă *real* și *effective GID*
- *setgroups*: stabilește grupuri suplimentare (necesită *effective UID = 0*)
- *initgroups*: alternativă la *setgroups* (necesită *effective UID = 0*)
- atenție
 - *effective UID = 0* \implies proces root, funcțiile din grup au comportament special
 - *effective GID = 0* \implies proces non-root

Utilizarea greșită a privilegiilor - descriere

- context
 - programe care rulează cu drepturi elevate
- greșeli
 - efectuează acțiuni potențial periculoase în numele unui utilizator cu mai puține privilegii, fără a-și micșora mai întâi privilegiile
 - nu se iau masuri de precauție înainte de interacțiunea cu sistemul de fișiere
- rezultat
 - expune fișiere importante \implies *information leakage*
 - sistemul este compromis
- sfat
 - un program *SUID* / *SGID* trebuie să-și coboare (temporar) privilegiile atunci când efectuează o operație interzisă pentru *real UID*
 - alternativă: verificați permisiunile pe baza *real UID*

Exemplu

- programul SUID root *XF86_SVGS*

```
$ id
uid=1000(test) gid=1000(test) groups=1000(test)

$ ls -l /etc/shadow
-rw----- 1 root root .... /etc/shadow

$ cd /usr/X11R6/bin

$ ./XF86_SVGA -config /etc/shadow
Unrecognized option: root:qEXaUxSeQ45la:10171:-1:-1:-1:-1:-1:-1
...
```

- programul citește fișiere fără a verifica dacă *real UID* are permisiuni pe acel fișier
- SUID* nu a fost folosit pentru a citi fișiere de configurare, a fost folosit pentru a afișa schimbările din fișierele de configurare

Biblioteci

- folosirea bibliotecilor externe
- sunt adesea sursa potențialelor probleme de securitate
- utilizatorii nu au detalii despre implementare, cunosc doar API

Scaderea permanenta a privilegiilor - context

- cod folosit pentru a micșora permanent privilegiile

```
// operatii cu privilegii elevate  
// configurare socket  
setup_socket();
```

```
// coborarea privilegiilor  
setuid(getuid());
```

```
// operatii neprivilegiate  
start_procloop();
```

- în unele situații nu este de ajuns (în care?)

Privilegiile grupului

- programe cu *SUID* și *SGID*
- **greșeala**: programul uită să-și scadă privilegiile grupului, pe lângă *UID*
- **greșeală**: ordinea greșită

```
// coborarea privilegiilor utilizatorului  
setuid(getuid());
```

```
// având privilegiile utilizatorului nou  
// nu se pot cobora privilegiile grupului vechi  
setgid(getgid());
```

- *saved SGID* poate să rămână în grupul privilegiat
- un atacator poate executa codul
`setegid(0);` sau `setregid(-1,0);`
pentru a recupera privilegiile grupului

Privilegiile grupului (II)

- ordinea corectă

```
// coborarea privilegiilor grupului  
setgid(getgid());
```

```
// coborarea privilegiilor utilizatorului  
setuid(getuid());
```

Supplemental Group Privileges

- context
 - procese pornite ca și utilizatori cu privilegii elevate, ex daemon
 - pornește cu privilegiile grupului = privilegiile utilizatorului
 - preia rolul unui utilizator neprivilegiat
- greșeala
 - aplicația își coboară privilegiile dar
 - lasa privilegiile suplimentare ale grupului utilizatorului privilegiat la utilizatorul cu mai puține privilegii
- incomplet, cod vulnerabil

```
if (root) {  
    setgid(normal_uid);  
    setuid(normal_gid);  
}
```


Supplemental Group Privileges (II)

- versiunea corectă pentru scăderea privilegiilor

```
if (root) {  
    setgroups(0, NULL);  
    setgid(normal_uid);  
    setuid(normal_gid);  
}
```

privilegii elevate pentru non-root

- context
 - un program *SUID* ce aparține unui utilizator non-root
- rulare ca și non-root

```
setgid ( getgid ( ));  
setuid ( getuid ( ));
```

- `setuid()` și `setgid()` schimbă doar *effective ID* nu și *saved ID*
 - atacatorii pot redobândi privilegiile aplicației
- soluție: folosiți următoarele funcții
 - `setresgid()` / `setregid()`
 - `setresuid()` / `setreuid()`

Combinarea între renunțarea temporară / permanentă a privilegiilor

- specific aplicațiilor care
 - schimbă între utilizatori cu diferite privilegii
 - eventual scăderea privilegiilor la toți utilizatorii (dacă este posibil)
- pot apărea erori datorită utilizării `setuid()`

Combinarea între renunțarea temporară / permanentă a privilegiilor - exemplu

```
void main_loop() {}
    uid_t realuid = getuid();
    // nu e nevoie de privilegii
    DROP_PRIV
    do_unprivileged_action();
    // primește inapoi privilegii
    RAISE_PRIV
    do_privileged_action();
    ....
    // nu necesita privilegii
    DROP_PRIV
    ...
    // coboara permanent privilegiile
    // !!! nu e root, -> SUID nu se schimba
    setuid(realuid);
    ...
}
```

- la scăderea permanentă a privilegiilor *effective UID* al procesului nu este 0 ci *realuid*, *saved SUID* rămâne neschimbat (ramâne 0)
- un atacator poate apela (dintr-un program compromis) "*seteuid(0)*:"_{44 / 54}

Scăderea temporară a privilegiilor

- abandonarea permanentă a privilegiilor este cea mai sigură opțiune pentru o aplicație setuid
- dar trebuie folosit apelul corect
- `seteuid(getuid());`
 - opțiune bună pentru scăderea temporară a privilegiilor
 - opțiune proastă pentru abandonarea permanentă a privilegiilor
- apelul corect pentru abandonarea permanentă a privilegiilor: `setuid(getuid());`
 - merge doar pentru root

Mai multe conturi utilizator

- specific programelor ce necesită mai mult de un cont de utilizator
- implementarea greșită

```
// devin user1  
seteuid (user1);  
process_log1 ();
```

```
// devin user2  
seteuid (user2);  
process_log1 ();
```

```
// devin root  
seteuid (0);
```

Mai multe conturi utilizator (II)

- implementarea corectă

```
// devin user1
seteuid (user1);
process_log1 ();
```

```
// devin root
seteuid (0);
```

```
// devin user2
seteuid (user2);
process_log1 ();
```

```
// devin root
seteuid (0);
```

Audit - scăderea permanentă a privilegiilor pentru procesele root

- ❶ când se scad privilegiile $UID = 0$
- ❷ stergerea grupurilor suplimentare
 - folosind *setgroups* cu *effective UID = 0*
- ❸ cele trei *GID* trebuie coborâte la un nivel *GID* neprivilegiat
 - abordare corectă: `setgid(getgid());`
 - abordare greșită: `setegid(getgid());`
- ❹ cele trei *GID* trebuie coborâte la un nivel *UID* neprivilegiat
 - abordare corectă: `setuid(getuid());`
 - abordare greșită: `seteuid(getuid());`

Audit - scăderea permanentă a privilegiilor pentru procesele non-root

- ❶ nu se pot modifica grupurile cu `setgroups()`
- ❷ pentru coborârea *GID* se foloseşte
 - `setresgid(getgid(), getgid(), getgid());`
- ❸ pentru coborârea *UID* se foloseşte
 - `setresuid(getuid(), getuid(), getuid());`

Limitarea sistemului de privilegii pentru sistemele UNIX

- modelul de privilegii "totul sau nimic"
- root are acces nerestricționat
- exemplu
 - ping are nevoie de acces root pentru a crea raw socket
 - dacă este exploatat înainte să-și coboare privilegiile, programul are acces total asupra resurselor sistemului
- orice program care necesită privilegii speciale pune siguranța întregului sistem în pericol

Bibliografie

- ① “The Art of Software Security Assessments”, chapter 9, “UNIX 1. Privileges and Files”, pp. 476 – 576
- ② “24 Deadly Sins of Software Security”, chapter 16, “Executing Code with Too Much Privilege”.