**Universitat de Barcelona**

Facultat de Matemàtiques i Informàtica

# Pràctica 1: Seguretat en els Sistemes Operatius

*Sistemes Operatius II*

Martí Martínez Gargallo i Víctor Sort Rubio

Professor: Oliver Díaz

# Contents

# 1 Introducció

L'objectiu d'aquesta primera pràctica consisteix en entendre i treballar amb aspectes sobre la seguretat dels sistemes operatius. A la primera part ens centrem en centrem en entendre els termes de propietatri, grup, permisos i alguna comanda com *chmod*. A la segona part de l'informe en canvi treballem sobre els buffers overflows i els problemes de seguretat que comporten.

# 2 Respostes i proves realitzades

## 2.1 Administrant la seguretat

### 2.1.1 Mitjançant línies de comandes, com pots mostrar els usuaris i grups del sistema?

Per a mostrar els usuaris usarem la següent comanda: *less /etc/passwd* on es pot veure una llista de tots els usuaris creats amb els seus GID i UID, així com el directori d'inici i la shell. Per exemple ens apareix:

```
oslabadmin:x:1000:1000:oslabadmin:/home/oslabadmin:/bin/bash}

oslab:x:1001:1001:,,,:/home/oslab:/bin/bash
```

Veiem que el GID i l'UID d'oslab és 1001, i el de oslabadmin és 1000. De manera similar, usant la comanda *less /etc/group* es pot veure una llista dels grups amb els seus GID.

### 2.1.2 Qué passa quan s'executa la comanda *su oslabadmin*? Per a qué serveix la comanda *su*? En quines situacions seria interessant utilitzarla? Qué succeeix si s'obre una nova terminal?

Quan s'executa la comanda *su oslabadmin* ens pregunta la contrassenya d'aquest usuari i es canvia a l'usuari oslabadmin durant tota la sessió de login. Si s'obre una nova terminal aquesta s'inicia en l'usuari que teníem des d'un principi.

Així doncs, aquesta comanda serveix per a canviar d'usuari durant una mateixa sessió de login, és a dir, sense tancar la sessió que estàvem utilitzant. Pot resultar interessant per a executar comandes en l'usuari root ja que és més còmode que la comanda sudo quan es realitzen més d'una comanda ja que només es demana la contrassenya al principi. Un inconvenient és que és un mètode menys segur.

### 2.1.3 Com podem veure les propietats del document creat (ejemplo.txt)? Qui és el propietari de l'arxiu? A quin grup pertany? Quins permisos té el propietari, grup i altres usuaris sobre aquest arxiu?

Per veure les propietats d'un document podem usar la comanda *ls -l ejemplo.txt*. Sen's retorna la línea:

```
-rw-rw-r-- 1 oslabadmin oslabadmin 25 sep 26 23:28 ejemplo.txt
```

El guió inicial ens indica que no és un directori, en cas contrari seria una d. Pels següents tres caràcters podem veure que el propietari té permisos de lectura i escriptura. Pels següents tres caràcters podem veure que el grup té també permisos de lectura i escriptura. Pels següents tres caràcters podem veure que els altres usuaris tenen només permisos de lectura.

A continuació veiem que el propietari és oslabadmin, i el grup al qual pertany és també oslabadmin. Això es deu a que estem encara estem a la sessió d'oslabadmin, doncs prèviament hem fet *su oslabadmin*.

#### 2.1.4 Qué succeeix a l'intentar guardar les modificacions a l'arxiu? Perqué?

No ens deixa guardar les modificacions a l'arxiu perquè no tenim els permisos d'escriptura, ja que no en som els propietaris ni estem al grup oslabadmin, com hem pogut observar a l'anterior pregunta.

#### 2.1.5 Pots veure el contingut de l'arxiu? Perqué? Quina acció s'ha realitzat amb la comanda *chmod 700 ./ejemplo.txt?*

No els podem veure, ja que amb la comanda s'han eliminat els permisos de lectura per altres usuaris. Utilitzem la següent taula per veure quins permisos s'han configurat.

| Permisos | Usuari | Grup | Altres |
|---|---|---|---|
| Lectura | 400 | 40 | 4 |
| Escriptura | 200 | 20 | 2 |
| Execució | 100 | 10 | 1 |

Com $700 = 400 + 200 + 100$ els permisos configurats són tots per l'usuari i cap pel grup i altres. (També ho podriem haver vist usant de nou la comanda *ls -l ejemplo.txt*)

#### 2.1.6 Quina comanda utilitzaries per canviar els permisos d'una carpeta per restringir l'accés a qualsevol que no sigui oslabadmin o que pertanyi al seu grup?

D'acord amb l'anterior taula, podriem usar la comanda *chmod 770 carpeta-propietats* que donaria tots els permisos al propietari oslabadmin i al grup, i cap a altres usuaris, ja que $770 = 400 + 200 + 100 + 40 + 20 + 10$. Ho testejem per veure que efectivament els permisos canvien i no podem accedir als seus continguts.

#### 2.1.7 Com canviaries els permisos per permetre l'execució de l'executable/script?

D'acord amb l'anterior taula, podriem usar la comanda *chmod 771 ./script.c* que donaria tots els permisos al propietari oslabadmin i al grup, i només d'execució a altres usuaris, ja que $771 = 400+200+100+40+20+10+1$. Ho testejem per veure que efectivament els permisos canvien i es permet l'execució.

### 2.2 Buffer Overflow

#### 2.2.1 Perquè apareix el missatge **** stack smashing detected ***: terminated Aborted (core dumped)*? Perquè apareix amb "SistemasOperativos" i no amb les altres contrassenyes? Per la contrassenya de "SistemasOperativos", com es podria evitar el core dumped?

En primer lloc quan compilem, ja ens apareix un warning perquè s'està actualitzant la funció gets (ens suggereix la fgets).

Aquest missatge d'error a l'executar apareix quan hi ha un desbordament de buffer, doncs pot causar problemes de seguretat al poder un atacant executar codi maliciós aprofitant aquesta vulnerabilitat.

Apareix amb la contrassenya "SistemasOperativos" i no amb la resta doncs el tamany de la contrassenya supera el del buffer, de només 15 caràcters.

Per evitar el core dumped es podria, a més d'augmentar el tamany del buffer, canviar la funcio gets per la funció fgets, doncs amb aquesta es pot controlar el nombre màxim de caràcters que es llegiràn i evitar un desbordament de buffer. Canviem doncs la linea amb la funció get per la següent:

```
fgets(buff, sizeof(buff), stdin)

buff[strcspn(buff, "/n")] = '/0'; //Elimina el caràcter de nova línea que agafa fgets
```

#### 2.2.2 Un cop deshabilitat el missatge de *stack smashing detected*, analitza el comportament del programa per les diferents contrasenyes anteriors i justifica el comportament.

El programa es comporta tal i com seria d'esperar. No es mostra error per cap contrassenya de més de 15 caràcters, doncs només llegeix els 15 primers com hem comentat abans. Clarament, per contrassenyes més curtes que no siguin "CafeConLeche" el programa segueix retornant que la contrassenya és incorrecte.

#### 2.2.3 Analitza vulnerabilitats al codi stack4.c i descriu el que fa l'exploit.

Inicialment, observem que el codi *stack4.c* utilitza el mètode gets en lloc del fgets, que com ja hem vist pot donar lloc a vulneracions de seguretat.

Quan executem la comanda ens torna a aparèixer un error com el d'abans (*** stack smashing detected ***), per tant tenim algun desbordament. El codi en Python exploit assigna en primer lloc a la variable 'LLETRA_A' una lletra 'A' en hexadecimal i, a continuació, la concadena 64 vegades per omplir el buffer de dades. A continuació afegeix tres blocs de 8 caràcters de A i amb la última comanda afegeix una direcció de retorn específica en hexadecimal.

L'objectiu del codi exploit és doncs construir una cadena d'entrada per aprofitar un desbordament de buffer en el codi *stack4.c* i redirigir l'execució a una direcció específica de memòria.

#### 2.2.4 Perquè no funciona amb python3 i si amb python2?

Python 2 permet tractar les cadenes de caràcters de manera més laxa, el que facilita la construcció d'exploits que aprofiten el desbordament del buffer. En canvi, Python 3 és més estricte amb les cadenes i dificulta els exploits.

En aquest cas concret, dóna un error pel print, ja que al Python 3 la funció print va seguida d'un parèntesi i en aquest cas no el té. Si fem el canvi dins el codi podem veure com funciona i dóna el mateix resultat.

#### 2.2.5 La direcció de la funció, s'ha de proporcionar en Big Endian o Little Endian? Quina és la diferència entre elles? Com es pot esbrinar si l'arquitectura de la màquina usa un o l'altre?

La direcció s'ha de proporcionar amb Little Endian. La diferència entre Big Endian i Little Endian és principalment l'ordre amb el que es guarden els bytes. En una arquitectura Big Endian, el byte de més valor s'emmagatzema a l'adreça de memòria més baixa. En canvi a Little Endian el byte menys significatiu s'emmagatzema a l'adreça més alta (al revés). Podem esbrinar-ho amb:

```
echo -n I | od -to2 | awk 'NR==1{print \$2}' | cut -c6
```

En el nostre cas, executant aquesta comanda ens dóna 1 de resultat que significa que la màquina utilitza Little Endian, que és doncs com haurem de proporcionar la direcció de la funció.

#### 2.2.6 Descriu el procés realitzat per explotar correctament el fitxer i que surti el missatge de *Congratulations, you've finished phoenix/stack-four :-) Well done!*

Per tant el que hem fet ha estat modificar la return address del fitxer "stack4_exploit.py" per la direcció de la funció "complete_level" que hem esbrinat amb:

```
readelf -s ./stack4
```

La direcció ha resultat ser (00000000000011c9) i en un principi l'hem posat tal qual a l'exploit. Ens ha seguit donant l'error, sense executar-se la funció que desitjàvem. Ens hem fixat que l'error era que estàvem posant la direcció en Big Endian. Per tant el que hem fet ha estat girar-la i ja teníem el procés finalitzat:

```
exploit += "\xc9\x11\x00\x00\x00\x00\x00\x00"
```

* Hi ha un possible error amb la vm, veure l'últim exercici.

#### 2.2.7 Què pots dir del rang de direccions i permisos de la pila?

En primer lloc destacarem les vulnerabilitats d'aquest nou programa "stack5". Aquestes tornen a aparèixer pel gets i el possible desbordament del buffer.

En relació a la preguntap destacarem que hem utilitzat *pgrep* per trobar l'ID del procés. Podem dir que el rang de direccions de la pila (stack) és 7ffcaf08f000-7ffcaf0b0000 i els seus permisos són "rw-p".

#### 2.2.8 Descriu el procés per desactivar les mesures de seguretat per a que la variable buffer de stack5.c tingui sempre la mateixa direcció de memòria i ens permeti escriure en ella.

Per a aconseguir-ho hem fet ús de dues comandes, les quals previamènt hem hagut d'instal·lar els seus respectius paquets. L'execució ha estat la següent:

```
> execstack -s ./stack5
oslab@oslab:~/Documents/p1-codigo$ setarch $(arch) -R ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7fffffffdfd0

> setarch $(arch) -R ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7fffffffdfd0
```

#### 2.2.9 Quin codi s'ha injectat? On està present aquesta injecció de codi en *stack5exploit.py*?

A continuació veiem l'execució, podem apreciar que les direccions són les mateixes, per tant hi ha un possible error amb la vm (stack canaries, shellcode, seguretat...). Per tant, no podem veure la injecció de codi, però aquesta s'injectaria tot seguida a la pila i el codi és el marcat al fitxer de python a la variable exploit (tots els NOP).

```
Buffer address: 0x7fffffffdfd0
and will be returning to 0x7fffffffdfd0
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

## 3 Conclusions i valoracions personals

Un cop finalitzada aquesta pràctica estem convençuts de que hem aconseguit assolir els coneixements que s'esperaven. Hem entés tots els conceptes relacionats amb la seguretat dels sistemes operatius explicats i la vulnerabilitat a la que es posa un SO al tenir buffer overflows, fent-lo susceptible de rebre atacs de codis maliciosos. Creiem a més a més que el nivell de dificultat de la pràctica no és excessiu i el temps invertit en la seva realització és correcte. Precissament, la part on hem tingut més problemes ha sigut a l'hora d'instalar-nos la màquina virtual donada i no a l'hora de resoldre la pràctica en si.

## 4 Contribució dels membres del grup

Per acabar aquest informe, comentar que hem treballat molt bé en equip, ens hem ajudat mútuament i ens hem repartit les tasques a fer de manera equitativa. Hem anat fent els exercicis alhora