Universidad de Barcelona

Arthur Font Gouveia 20222613

Marc Colominas Casabella 20081692

Sistemes Operatius II

Pràctica 2

Barcelona

2021

Índex

1. Introducció

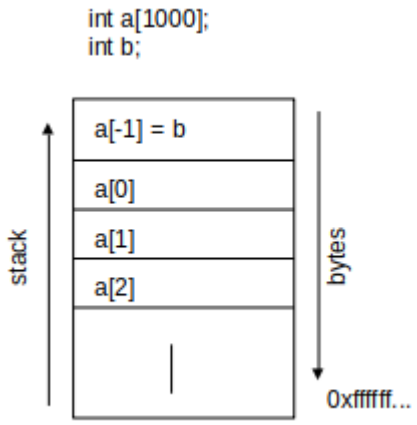
En aquesta pràctica farem una breu aproximació al concepte de ciberseguretat i ens centrarem en algunes tècniques que es poden utilitzar per aprofitar els buffer overflows així com altres vulnerabilitats de corrupció de memòria.

2. Stack overflow

2.1. Sobreescriptura de la direcció de retorn

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? Feu-vos un dibuix perquè us quedi clar ja que ara en traurem profit!

S'ha pogut modificar el valor de b perquè la posició de memòria que correspón al valor de a[-1] és el mateix que correspón al valor de b. La pila emmagatzemen les variables en sentit decreixent en la memoria.



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?

Direcció de retorn: 0x4005e7

Hem modificat l'exploit per tal de que la direcció de retorn apunti a la funció complete_level. S'ha de tenir en compte que la arquitectura del ordinador, en este cas la arquitectura utilitza el Little endian, on el bytes d'un registre s'emmagatzemen a memòria RAM de menys a més significant.

2.2. 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 just després de l'últim byte del buffer, on estará localitzat el shellcode.

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?

Es pot observar que les direccions de la majoría de les zones canvien, excepte la direcció de [vsyscall] i les direccions on es troba el codi màquina. La pila i les llibreries dinàmiques es carreguen al fi de la memòria virtual.

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, perquè les direccions de memòria es canvien aleatòriament a cada vegada que s'executa un nou procés per qüestions de seguretat. Més a més, la pila no té permís d'execució, com podem observar amb la comanda cat */proc/<process_ID>/maps*.

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

No, perquè sería imposible saber quina seria la següent direcció de retorn. A més la pila no té permís d'execució.

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 el que fa és desactivar l'aleatorització de l'espai d'adreces virtuals. Serà molt útil per fer que la direcció de retorn apunti al començ del buffer.

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

La injecció del codi executa la comanda ls.

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?

Els bytes del codi que indiquen la instrucció són els 2f 62 69 6e 2f 6c 73. La instrucció assemblador que conté el codi a executar és el movq.

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?

Hem modificat el codi injectat perquè executi la comanda /bin/ps, per això hem canviat el valor 12º byte de 6c per 70, els valors corresponen a les lletres 'l' i 'p' respectivament.

```
oslab:/media/sf_shareVM/codi> python stack5_exploit.py | ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.
Buffer address: 0x7fffffffddc0
and will be returning to 0x7fffffffddc0
  PID TTY          TIME CMD
 2319 pts/0    00:00:00 bash
 2472 pts/0    00:00:00 bash
 2559 pts/0    00:00:00 bash
 2810 pts/0    00:00:00 ps
oslab:/media/sf_shareVM/codi>
```

3. Heap overflow

3.1 Sobreescriptura de la direcció de retorn

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

S'ha d'escriure a name de i2 la direcció de winner, que és 0x400637. Hem aconseguit obtenir la direcció de winner amb la comanda *readelf -s heapone* | *grep winner.*

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

S'ha d'escriure la direcció en què s'emmagatzema la direcció de retorn de la funció. Hem aconseguit obtenir la direcció amb la informació mostrada per pantalla buscant la adreça de la pila que conté el valor 0x400637 (direcció de winner).

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

No, perquè el shellcode injectat canvia directament la direcció de retorn a través de el buffer overflow de la funció strcpy.

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

Hem canviat el valor de A perquè sigui l'adreça on es guarda la direcció de retorn. i el valor de B perquè sigui l'adreça de la funció winner. Hem assignat els seguents valors per A i B:

A=$(python -c 'print "A" * 40 + "\x08\xde\xff\xff\xff\x7f"')

B=$(python -c 'print "\x37\x06\x40"')

Hem fet servir la comanda *readelf -s heapone* | *grep winner* per trobar l'adreça de la funció winner i hem mirat la informació enseñada en la pantalla per trobar l'adreça es guarda la direcció de retorn.

```
oslab:/media/sf_shareVM/codi> ./heapone_exploit.sh
Welcome to phoenix/heapone, brought to you by https://exploit.education
A la direccio de la pila 0x7fffffffddc8 emmagatzema el valor: 0x7ffff7dce500
A la direccio de la pila 0x7fffffffddd0 emmagatzema el valor: 0x7ffff7dce500
A la direccio de la pila 0x7fffffffddd8 emmagatzema el valor: 0x7fffffffddd8
A la direccio de la pila 0x7fffffffdde0 emmagatzema el valor: (nil)
A la direccio de la pila 0x7fffffffdde8 emmagatzema el valor: 0x4ffffde20
A la direccio de la pila 0x7fffffffddf0 emmagatzema el valor: (nil)
A la direccio de la pila 0x7fffffffddf8 emmagatzema el valor: (nil)
A la direccio de la pila 0x7fffffffde00 emmagatzema el valor: 0x7fffffffde20
A la direccio de la pila 0x7fffffffde08 emmagatzema el valor: 0x4007e5
A la direccio de la pila 0x7fffffffde10 emmagatzema el valor: 0x7fffffffdf18
Original return address: 0x4007e5
A i2->name s'emmagatzema el valor 0x7fffffffde08
New return address: 0x400637
and that's a wrap folks!
Congratulations, you've completed this level!!!
oslab:/media/sf_shareVM/codi>
```

4. Conclusions

En aquesta pràctica hem vist com funcionen certes tecniques per aprofitar els buffers overflows i altres vulnerabilitats de corrupció de memòria.

Hem vist com pot modificar un atacant la direcció de retorn de la pila, com programar un shellcode i també com modificar la direcció de retorn en un heap.

Tot i tenir certes dificultats amb l'apartat 3.1 finalment hem completat totes les parts de la pràctica correctament.