ShellCodes Paper

Darío Gutiérrez

Universidad Nacional de Quilmes
Buenos Aires, Argentina
dariofg93@gmail.com

En el presente informe, daré una breve definición de los Shellcodes, técnicas de implementación y un ejemplo muy simple para ver su práctica.

In this report, I will give a brief definition of the Shellcodes, implementation techniques and a very simple example to see their practice.

I. Que significa shellcode?

Primero es necesario conocer el significado etimológico de Shellcode.

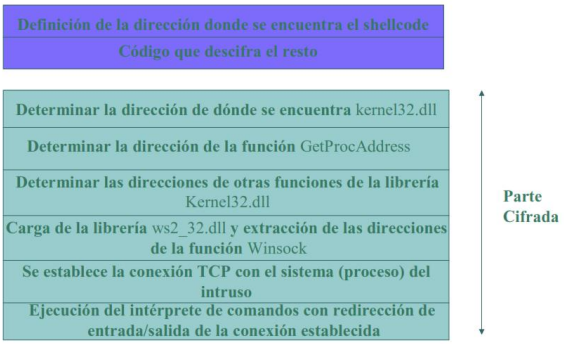
Shell es como se dice en inglés (ahora universal) a una terminal. Por lo tanto, shellcode significa que es un código que nos crea una terminal para que la usemos.

Actualmente el termino shellcode no solo se refiere al código que nos permite crear una Shell, sino que se extiende a códigos con más funciones como pueden ser crear usuarios, matar/crear un determinado proceso, dar permisos a una carpeta, etc.

Y técnicamente… una shellcode es una porción de código compilada para un determinado procesador que se ejecuta dentro de otro proceso previamente preparado para que se ejecuta (ej.: stack overflow).

A. *Esqueleto Base del codigo Shell*

- **Obtener el EIP:** Direcciones base. Cualquier variable o función en el código Shell será relativa a esta dirección. Para obtener esta dirección utilizamos las funciones CALL y POP.
- **Decodificar:** Generalmente encontramos en memoria caracteres NULL, los cuales nos impiden ejecutar nuestro código. Lo que implica que debamos codificar nuestro código generalmente con XOR con un valor predefinido.
- **Obtener las direcciones de las funciones requeridas:** Identificar en memoria los API, para identificar las direcciones de los procesos que requerimos.
- **Configurar el Socket de conexión:** Generalmente se requiere ubicar donde están (en memoria) las funciones de socket (), bind (), listen () o sus equivalentes.
- **Creación del Shell**



Cabe resaltar que el nivel del shellcode también se rige por el algoritmo que se utiliza para la ofuscación o cifrado del código, eso invoca diseñar un algoritmo propio de cifrado, muy independiente de los que existen en la actualidad.

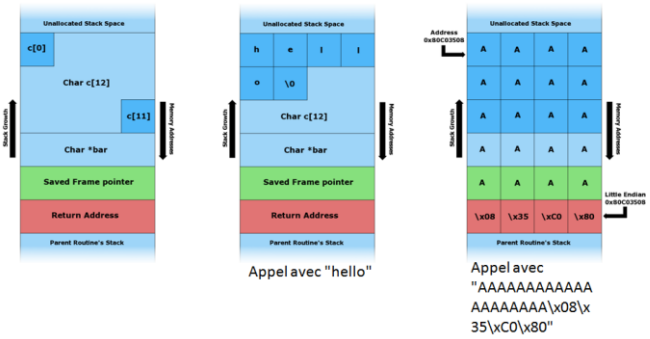
II. Considereaciones de funcionamiento

Las ShellCodes son generalmente utilizadas para explotar vulnerabilidades, cuyas técnicas empleadas son, entre otras:

A. *Stack Buffers Overflows.*

Hay que tener presente que se origina por la manipulación o mal uso de buffer (Conjunto contiguo de espacio de memoria).

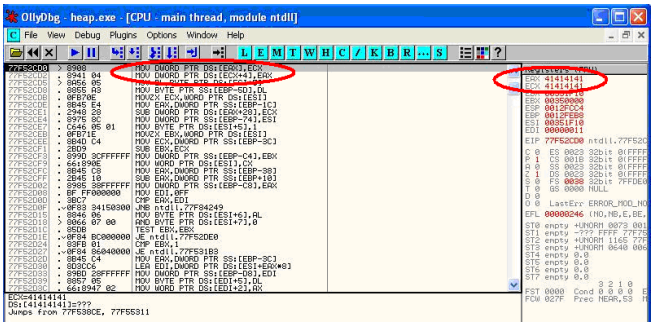
Stack Buffer Overflow (Desbordamiento de pila) se producen cuando los datos de tamaño variable se copian en buffers de longitud fija situados en la pila del programa sin ningún tipo de comprobación o límites.



Vulnerabilidades de desbordamiento de pila a menudo permiten a un atacante tomar el control directo del puntero de la instrucción y, por tanto, modificar la ejecución del programa y ejecutar código arbitrario. Además de sobrescribir el puntero de instrucción, resultados similares también pueden obtenerse a través de sobrescribir otras variables y estructuras, como manejadores de excepciones, que se encuentran en la pila.

B. *Heap Buffers Overflows.*

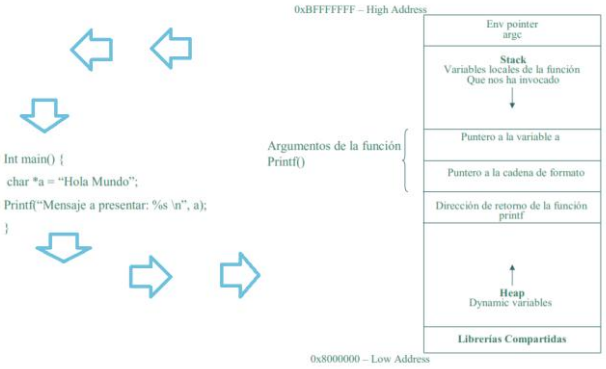
Mientras que en el caso de un desbordamiento de pila un puntero de instrucción o escritura SEH seria aparente, esto no es cierto para una condición de desbordamiento de Heap. Cuando se depura un programa de Windows, un desbordamiento de Heap puede aparecer en varias formas, la más común es un cambio de puntero que tiene lugar después de la rutina de gestión de Heap entra en acción. A continuación vamos a ilustrar una vulnerabilidad de desbordamiento de Heap.



Los dos registros mostrados, EAX y ECX, se pueden rellenar con dirección de usuarios que forman parte de los datos que se utilizan para desbordar el buffer suministrado. Cuando se ejecuta las instrucciones MOV mostradas en el panel de la izquierda, el sobrescribir tiene lugar y, cuando se llama a la función, el código de usuario suministrado se ejecuta.

C. *Format String.*

El exploit para String Format ocurre cuando los datos presentados de una cadena de entrada se evalúan como una orden por la aplicación. De esta manera, el atacante podría ejecutar código, leer la pila, o causar un fallo de segmentación en la aplicación en la ejecución, provocando nuevos comportamientos que podrían comprometer la seguridad o la estabilidad del sistema.



III. COMO CREAR UNA SHELLCODE?

Generalmente se hacen en lenguaje ASM (assembler) para poder tener un control total del proceso ya que en C se agregan partes del código que no podemos controlar, ni hablar de otros lenguajes que ni siquiera pueden compilarse a código natico del procesador (.NET, VB, Java, etc.).

Asm de x86(el procesador que usan las pcs) tienen dos sintaxis, la de Intel y la de AT&T, usaremos la primera.

Se sabe que una shellcode debe ser lo más corta y portable posible, es por eso que se usara llamadas al sistema y no llamadas a la libc u otras librerías. La forma de llama al sistema/kernel en *nix es mediante la interrupción 0x80.

A. *Código ASM*

```
1. BITS 32
2. exit:
3.    xor ebx,ebx
4.   ;ponemos 0 como argumento de exit
5.    Xor eax,eax
6.   ;limpiamos eax y ebx
7.    mov al,0x01
8.   ;ponemos 1 en eax (es el código
   de la syscall exit)
9.    Int 0x80 ;finalmente llamamos a
   la interrupción
```

B. *Extraer los bytes u opcodes*

A través de un compilador se extraen los bytecodes. Se utilizara nasm (en Linux se puede instalar al abrir una consola y tipeando sudo apt-get install nasm) y por eso abrimos una terminal y tipeamos:

~$ nasm exit.asm (nombre del archivo assembler)
Para saber si se compilo se desensambla (en 32 bits)
~$ ndisasm-b 32 exit
Desensamblado del código con sus opcodes:

00000000 31DB xor ebx,ebx
00000002 31C0 xor eax,eax
00000004 B001 mov al,0x1
00000006 CD80 int 0x80

C. *Cadena C para el exploit*

Para sacar estos opcodes podemos copiarlos uno por uno al exploit o a donde se quieran poner; o extraerlos con alguna clase script, se utilizara uno (en Python) que pasa el archivo a la cadena de C directamente:

```
1.  #!/usr/bin/python
2.  from binascii import *
3.  import sys
4.
5.  files = {}
6.  length = []
7.
8.  def write(string):
9.      sys.stdout.write(string)
10.
11. def process_file(file):
12.     source = open(file,"rb")
13.     write("char shellcode[]=\n\t\t")
14.     i = 1
15.     l = 0
16.     write("\"")
17.     for data in source.read():
18.         write("\\x")
19.         write(b2a_hex(data))
20.         if (i % 15)==0:
21.             write("\"\n\t\t\"")
22.         i+=1
23.         l+=1
```

```
24.     files[file[0:-4]]= l
25.     write("\";\n\n")
26.     source.close()
27.
28. print("HDL Shellcode Lab - Bin to C")
29. if(len(sys.argv)<2):
30.     exit(0)
31. process_file(sys.argv[1])
```

Como argumento se le pasa el archivo compilado anteriormente con nasm y devuelve una cadena en C para poner en nuestro exploit.

D. *Como probar las Shellcodes?*

Para probarlo se usa un pequeño código en C que salte a la shellcode directamente.

```
1.  char shellcode[] =
    "\x31\xdb\x31\xc0\xb0\x01\xcd\x80";
    //acá ponemos la shellcode
2.
3.  int main (int argc, char *argv[])
4.  {
5.    int (*run)();
6.  //puntero a una función
7.    run = shellcode;
8.  //asignamos la dirección de la
    shellcode al puntero
9.    run();  // y se ejecuta
10. }
```

Links and Bookmarks

Toda la información fue obtenida de distintos websites [1], [2], [3].

Para más información sobre exploits [4].

IV. Conclusion

Hacer una Shellcode es algo que todo programador debería en algún momento aprender a hacer, así como también los recursos y las ventajas que un lenguaje tan poderoso como el ASM tiene.

El reto de crear una Shellcode y hacerlo funcionar como uno quiere es motivo de orgullo y un aprendizaje muy valioso para la carrera de uno.

Acknowledgment

A toda la comunidad que investiga y comparte sus conocimientos y a los servicios de internet que se ocupan de hacer llegar la información.

References

[1] https://underc0de.org/foro/hacking/que-es-una-shellcode/ Post de Usuario ANTRAX, website: Undercode, Mzo. 2012.

[2] http://antisec-security.blogspot.com.ar/2013/08/introduccion-shellcode-bueno-en.html Post de Usuario J@lil K@mel, website: Seguridad informática, agto. 2013.

[3] https://www.exploit-db.com/papers/35646/ Paper publicado por contacto: flor_iano@hotmail.com website: Exploit Database, actualizado 2017.

[4] https://es.wikipedia.org/wiki/Exploit Artículo de Wikipedia, website: Wikipedia, actualizado feb. 2017.