Api Hooking by MazarD

1.-Introducción

El api hooking es un tema que me ha apasionado desde que he empezado con él. Imagina la potencia que dá el poder saber cuando cualquier programa ha llamado a cierta api e incluso poder modificar el resultado que va a recibir. Por dar algunos ejemplos se pueden ocultar procesos, ocultar arhivos, esnifar tráfico de red, modificar el aspecto de cualquier ventana, espiar las conversaciones del messenger (te suena msnNightmare? xD)...
En resumen, control total de nuestro sistema, los límites los pone nuestra imaginación.
En éste texto se explicará el método trampolín en userland para hookear la api MessageBoxExA y para llegar a comprender el texto debes tener conocimientos como mínimo básicos sobre asm, programación en win32 y sobretodo de c.
Para hookear cualquier api además tienes que saber moverte con soltura delante de un debuguer.

2.-Método de inyección

Para poder hookear la api necesitamos que nuestro código esté en su mismo espacio de memória, para poder realizar saltos desde api a dll, dll a api. Por lo tanto necesitaremos inyectar código en el ejecutable que la utilice, para esto hay unos cuántos métodos, desde modificar la libreria que contiene la api en disco hasta usar createremotethread e inyectar directamente el código.
Éste es un tema demasiado extenso que necesitaría de otro tutorial así que para hacer las pruebas utilizaremos el siguiente programa que cargará nuestra dll y mostrará un MessageBox que deberemos hookear:

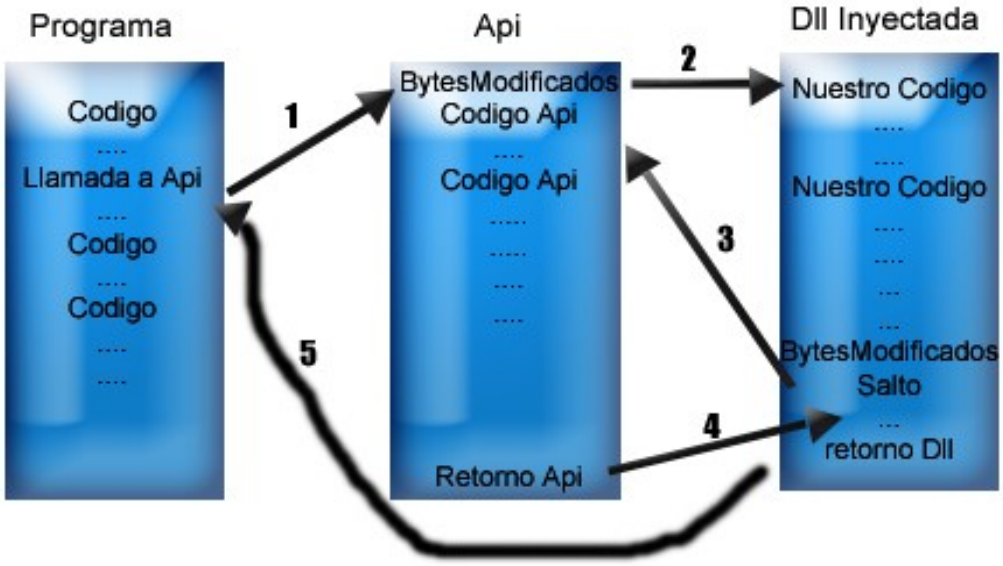
```
#include <windows.h>
#include <stdio.h>

int main(void)
{
	LoadLibrary("path\\a\\nuestra\\dll.dll");
	MessageBoxEx(NULL,"Soy un messagebox NO hookeado","ApiHooking",0,0);
	return 0;
}
```

3.-Algo de teoría

Nos vamos de caza.
Victima: MessageBoxExA
Arma: inyección dll
Objetivo: Sólo dirá lo que nosotros queramos que diga

El proceso es el siguiente:
Partimos de la base de que con tu inyección el programa tendrá cargada nuestra dll, con lo que ya queda claro que estamos en el espacio de memória de nuestra victima.
En el momento en que un programa se ejecuta carga nuestra dll y en el entry point (el código que se ejecuta sólo con cargar una dll) lo que haremos será cambiar los primeros bytes de la api para que salte a nuestro código, con esto conseguimos que cada vez que un programa llame a la victima en realidad nos está llamando a nosotros.
Ahora ya sólo devolviendo al programa principal los resultados que nosotros quisieramos habríamos cazado la api.
Eso sí, como hemos petado a la victima nosotros tampoco sabemos cual seria el resultado ni sabemos cómo emular la funcionalidad de la api, y al intentar hacer una llamada a ella estamos saltando de nuevo a nosotros.
Para solucionar esto lo que se hace es desde nuestro código ejecutar los bytes que nos habíamos cargado en la api original y saltar a la parte que continuaba de la api.
De éste modo conseguimos ser muy sigilosos con la api, es cazada pero no se dá cuenta. Mientras, el programa principal cree que nosotros somos la api ya que además podemos actuar cómo la victima pero modificando el comportamiento concreto que nos interese.



Sin tener en cuenta lo feo que me ha quedado el dibujo vemos cómo inicialmente el programa llama a cierta api, ésta tiene su código inicial modificado para que salte a nuestro código, haremos lo que nos dé la gana, ejecutaremos el código inicial que habíamos sobreescrito, saltaremos al punto siguiente a los bytes modificados, es decir, a la api original, esta nos devolverá la ejecución, trataremos sus resultados y devolveremos la ejecución al programa principal con los resultados que nos dé la gana.

Cuantos bytes hay que modificar del inicio de la api?
En principio siempre 5.
En el inicio de cualquier función se establece lo que se llama el enlaze dinámico, que vendría a ser un modo de establecer el trozo de pila que nos corresponde (push ebp,mov ebp,esp) esto no llega a los 5 bytes así que faltaría otra instrucción que puede variar dependiendo de la api.
Lo esencial es que se puedan ejecutar esas instrucciones en cualquier lugar en el código y que no se parta por el medio al cojerlos. Por poner un ejemplo **no** podríamos copiar sólo los 5 primeros de una api que empezara por:

833D C4D3D677 00 CMP DWORD PTR DS:[77D6D3C4],0

Puesto que los primeros 5 bytes serían 833DC4D3 y esto no sabemos a qué instrucciones corresponderán pero petará fijo, hay que tener mucha punteria con esto, ya que es lo que suele dar problemas.
Este es el caso de MessageBoxA, en lugar de copiar los 7 bytes utilizaremos la MessageBoxExA que sí tiene 5 bytes justos para no complicarnos la vida. Además si en lugar de un cmp tubieramos un salto no bastaría con copiar la instrucción y se debería recalcular la distancia lo que ya complicaría mucho la história. La mayoría de apis tienen los 5 bytes justos mencionados y sin necesidad de recalcular nada pero él único modo de asegurarse es meter la cabeza en el debugger y hechar un vistazo.
Además todavía nos quedan dos pequeños inconvenientes de menor importancia, el primero es que la victima tiene una pequeña armadura que no nos permitirá escribir en su espacio de memoria, pero lo vamos a solucionar rápidamente usando el api VirtualProtect, el segundo es que es posible que el exe en concreto tenga la librería de la api en caché con lo que nuestra inyección no tendría efecto, esto lo solucionaremos con FlushInstructionCache.

4.-Código de la dll

Ahora ya sí, vamos a lo que importa: PROGRAMAR, SEA LO QUE SEA, SEA CUANDO SEA

```
//Interceptación de apis by MazarD (API HOOKING)
//Hook a MessageBoxExA
//www.mazard.info
//MazarD@gmail.com
#include <windows.h>
#include <stdio.h>
//Puntero al buffer donde guardaremos las instrucciones copiadas de la api y
//el salto a la api+5
BYTE *Buffer;
//Ésta es la dll donde reside la api MessageBoxExA
char Libreria[]="user32.dll";
//El api en qüestión
char NomApi[]="MessageBoxExA";
//Funcion a la que llamará el programa principal creiendo que es la api original
int WINAPI FuncioRep(HWND hWnd,LPCSTR lpText,LPCTSTR lpCaption,UINT uType,UINT
LangId);
void Hookear(void);
//Puntero a función, utilizando este puntero conseguiremos ejecutar el código contenido en el buffer
int (__stdcall *pBuff)(HWND hWnd,LPCSTR lpText,LPCTSTR lpCaption,UINT uType,UINT
LangId);
//La función de reemplazo explicada arriba
int WINAPI FuncioRep(HWND hWnd,LPCTSTR lpText,LPCTSTR lpCaption,UINT uType,UINT
LangId)
{
```

```
//Cadena que le pegará vacilada al programa principal xD
char texto[]="Te acabo de hookear mwahahahaha";
int Resultado;
//Hacemos la llamada al puntero a buffer (api original) cambiandole el parametro texto
Resultado=pBuff(hWnd,texto,lpCaption,uType,LangId);
//Le damos el resultado que daría la api original al programa principal
return Resultado;
}
void Hookear(void)
{
DWORD ProteVieja;
BYTE *DirApi;
BYTE *DirYo;
//Cojemos la dirección de memoria de la api
DirApi=(BYTE *) GetProcAddress(GetModuleHandle(Libreria),NomApi);
//Creamos 10bytes de memoria para nuestro Buffer
Buffer=(BYTE *)malloc(10);
//Le damos todos los permisos a los 10 bytes de nuestro Buffer
VirtualProtect((void *) Buffer, 12, PAGE_EXECUTE_READWRITE, &ProteVieja);
//copiamos los 5 primeros bytes originales de la api antes de machacarlos
memcpy(Buffer,DirApi,5);
Buffer+=5;
//En el sexto introducimos E9 que corresponde a jmp(salto
//incondicional) en código máquina para que salte a la api original
*Buffer=0xE9;
Buffer++;
//A partir del septimo metemos 4 bytes que determinan la distancia del salto
//desde el buffer hasta la Dirección de la api+5
*((signed int *) Buffer)=(DirApi+1)-Buffer;
//Asignamos al puntero a funcion pBuff el inicio del Buffer
pBuff = (int (__stdcall *)(HWND,LPCTSTR,LPCTSTR,UINT,UINT)) (Buffer-6);
//Le damos todos los permisos a los 5 primeros bytes de la api original
VirtualProtect((void *) DirApi,5,PAGE_EXECUTE_READWRITE,&ProteVieja);
//Cambiamos el tipo a puntero a byte para facilitar el trabajo
DirYo=(BYTE *) FuncioRep;
//En el inicio de la api metemos un salto incondicional hacia nuestro código
//E9=jmp
*DirApi=0xE9;
DirApi++;
//Los 4 siguientes bytes determinan la distancia del salto desde la api hasta
//la función de reemplazo en nuestro código, fijate que en este caso el
//resultado tiene que ser negativo, puesto que las apis corren en direcciones
//de memória mucho mas altas y el salto deberá ser "hacia atrás"
*((signed int *) DirApi)=DirYo-(DirApi+4);
//libermos las librerias de cache
FlushInstructionCache(GetCurrentProcess(),NULL,NULL);
}
bool WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
//si se ha entrado en Dllmain porque se acaba de cargar la librería hookeamos
if (fdwReason == DLL_PROCESS_ATTACH) {
Hookear();
}
```

return TRUE;
}

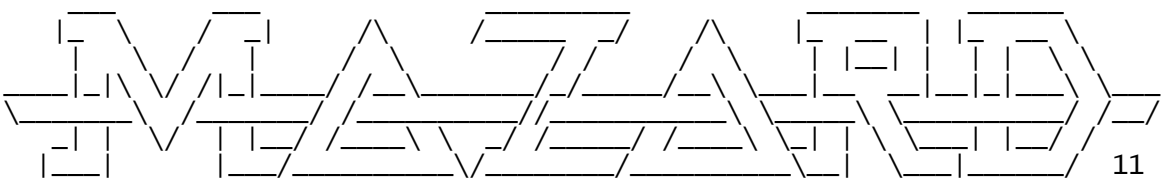
Cómo vemos una dll es cómo cualquier ejecutable, la única diferencia es que DllMain puede ser llamada por varios motivos, el que nos interesa sería cuando algun ejecutable nos carga, aunque también podría ser útil hacer nuestros trapicheos en DLL_PROCESS_DEATTACH.

Hay que fijarse que en el puntero a función para el buffer de los bytes reemplazados está usando la convención __stdcall esto me estubo dando problemas al principio ya que tenia un fallo de convenciones, si no se declara así los compiladores consideran que es __cdecl entonces no se sacan los parametros de la pila, el programa principal al ser una api tampoco los saca y termina petando.

5.-Despedida

Bueno, aquí termina esto, hemos cazado a MessageBox, espero que no te haya sido muy dura la lucha, la gracia está en que con ése mismo código cambiando los parámetros de las funciones y recalculando los saltos si el número de bytes iniciales de la api varia ya lo tienes solucionado. Lo demás ya sólo es ver cómo funciona la api que quieres atacar.

Para cualquier duda, suggeréncia, bug, optimización o chorrada:



YES! I walked through the dark ways of madness. And I´m still alive!!
MazarD [arroba] gmail [punto] com
http://www.mazard.info