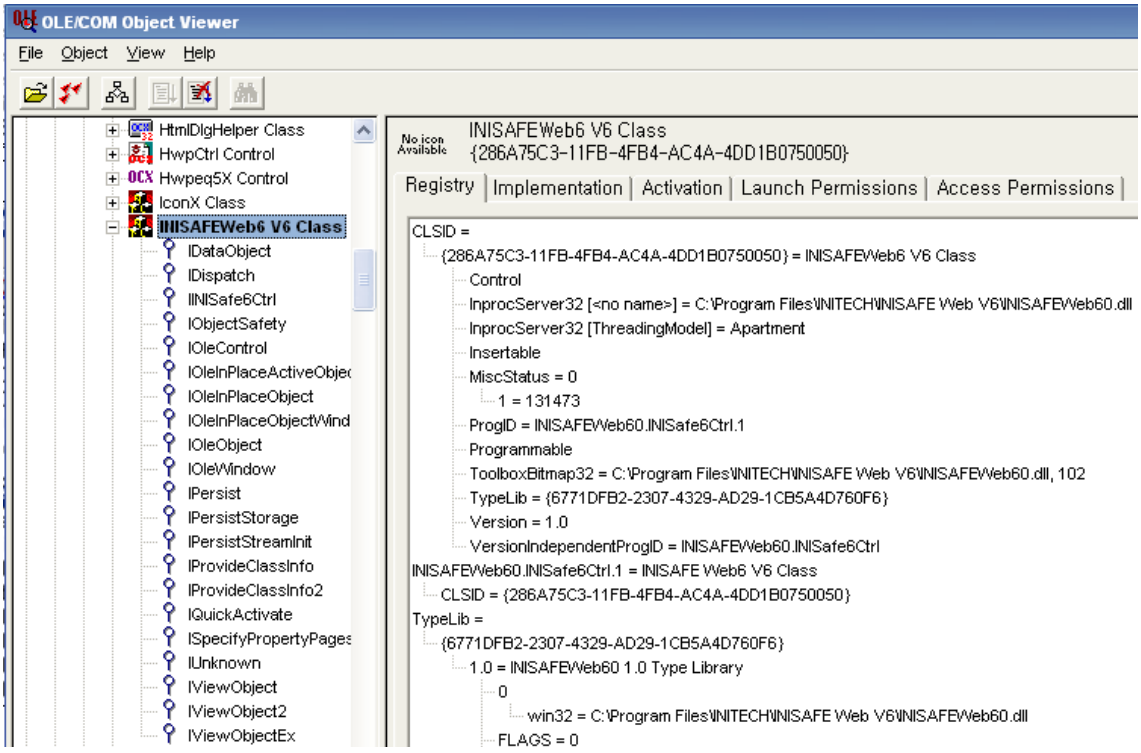
CODE Intercept Hoocking – DLL injection
INISAFEWeb6 보안모듈 Hoocking 하기

수원대학교 컴퓨터학과 flag 지선호(kissmefox@gmail.com)

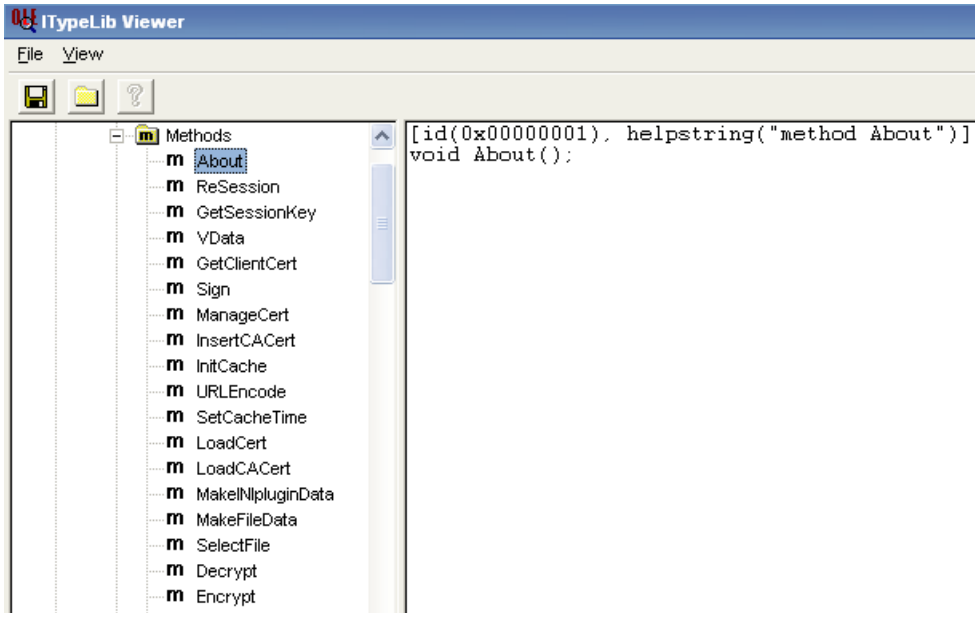
*AmesianX 님의 Art of Hooking 문서를 읽고 실습해 본 내용을 정리하였습니다
후킹의 종류는 API 후킹, 메시지 후킹, 코드 인터셉트 후킹 등의 종류로 나눌 수 있다. 여기서는 코드 인터셉트 후킹을 통해 목적 프로그램의 dll 에 악의적인 목적으로 제작된 dll 을 삽입하여 프로그램의 로직 자체를 변경시키는 과정을 진행해보려고 한다. 해킹 프로그램은 디아블로 2해킹 프로그램으로 제작된 D2HackIt 의 소스를 이용하여 공격을 진행할 것이며 목적 프로그램은 현재 금융기관이나 각종 온라인 게임, 인터넷 쇼핑몰 등에서 보안을 위해 사용되는 INISAFEWeb6 모듈을 타겟으로 진행할 것이다.

프로그램이 실행되면 먼저 임포트 테이블의 라이브러리(DLL)을 순서대로 하나씩 로딩하게 되는데 DLL 이 로딩 될 때는 DllMain 이라는 엔트리(Entry) 함수부터 호출이 된다. 즉, 우리가 DLL 프로그래밍을 할 때 프로그래밍 도입부였던 DllMain부터 코드가 시작될 것이라는 의미다. 여기서 순서대로라고 말한 이유는 먼저 로딩된 D2HackIt.dll 이 아직 로딩되지 않은 DLL 의 함수를 호출하게 되면 프로그램이 박살난다. 이런 로딩 순서를 지키기 위해서 때로는 EXE 파일에 D2HackIt.dll 을 박지 못하는 경우도 생길 수 있으며 EXE 파일 이외에 같은 디렉토리의 다른 DLL 파일에 박아야 하는 경우도 생길 수 있다.

금융거래에 관련된 결제를 수행하게되면 INISAFE 라는 모듈이 시스템이 자동으로 설치되어 있는 것을 확인할 수 있다.



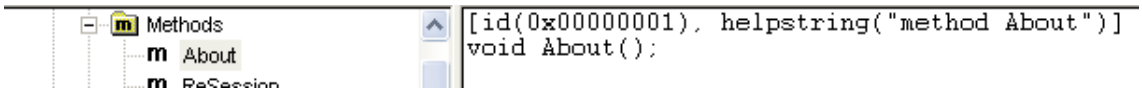
제공되는 함수들의 목록에서 about 함수를 호출해보자.



먼저 클립보드로 모듈의 클래스 아이디와 object 태그를 복사한뒤 실습을 위한 기본 HTML 태그 속에 삽입한다.

```
1 <HTML>
2 <HEAD>
3 <TITLE> inisafeTest </TITLE>
4 </HEAD>
5 <BODY>
6 <object
7    classid="clsid:286A75C3-11FB-4FB4-AC4A-4DD1B0750050"
8 >
9 </object>
10 </BODY>
11 </HTML>
```

호출될 함수의 프로토타입을 확인한 뒤

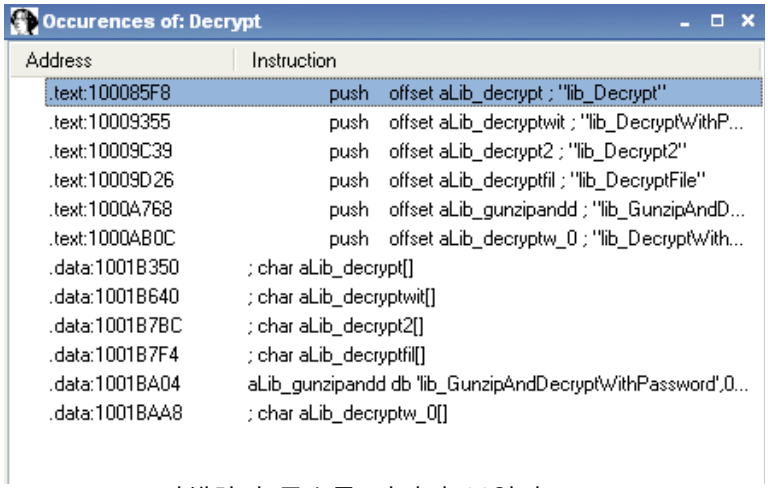


객체에 이름을 주고 자바스크립트로 객체의 about 함수를 호출해준다.

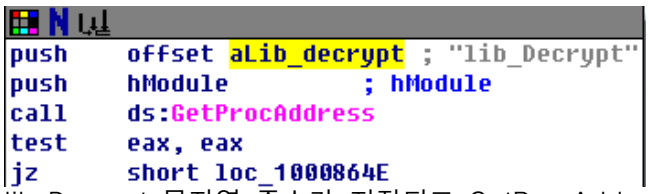
```
1 <HTML>
2 <HEAD>
3 <TITLE> inisafeTest </TITLE>
4 </HEAD>
5 <BODY>
6 <object name="test"
7    classid="clsid:286A75C3-11FB-4FB4-AC4A-4DD1B0750050"
8 >
9 <script>
10 document.all.test.About();
11 </script>
12 </object>
13 </BODY>
14 </HTML>
```

함수가 정상적으로 호출된 것을 확인할 수 있다.
이것으로 시스템에 설치된 COM 모듈은 클래스 아이디(CLSID)만 알면 어떤 프로그래밍 언어를 사용하더라고 호출이 가능하다는 것을 확인할 수 있다.

이제 이 보안모듈에서 가장 중요한 암호화, 복호화 함수인 Decrypt, Encrypt 함수를 후킹하여 정상적인 통신과정에서 함수의 인자들을 가로채어 그 인자들을 빼돌리는 과정을 수행해야 한다. IDA 로 해당 모듈의 DLL 파일을 디스어셈블링 한다.



Decrypt 로 검색하여 주소를 따라가 보았다.



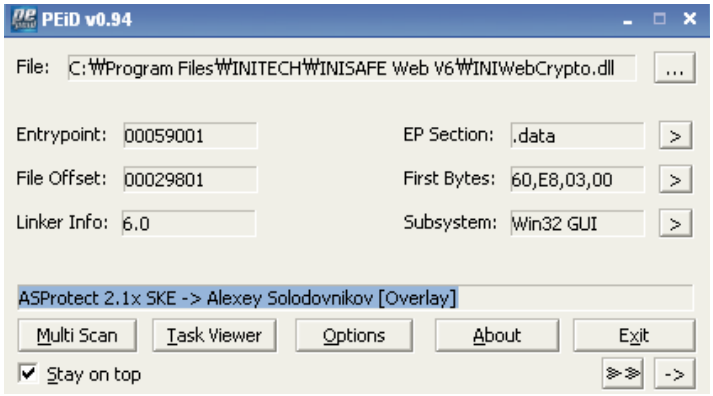
lib_Decrypt 문자열 주소가 저장되고 GetProcAddress 함수가 바로 뒤따라 나오는 걸로 봐서 다른 DLL 파일에 Decrypt 함수가 있다는 것을 알 수 있다.

lib_Decrypt 문자열을 따라가서 그 위치에서 위쪽을 검색하면 함수가 저장된 DLL 을 확인할 수 있다.

```
.data:1001B22C aIniwebcrypto_d db 'WINIWebCrypto.dll',0 ; DATA XREF: sub_10007F07+36↑o
.data:1001B23E                 align 10h
```

실제함수가 저장된 DLL 파일을 로드하였지만 패킹되어 정보가 제대로 출력이 되지 않았다.

패킹 툴을 확인해보니 언패킹하기 까다로운 ASProtect 2.1x SKE 였다..ㅠㅠ



MUP 도전 2시간..
언패킹툴 검색 2시간...
대부분의 언패킹 관련 기술들은 중국사이트에만 기술되어 있었다..
to be continue

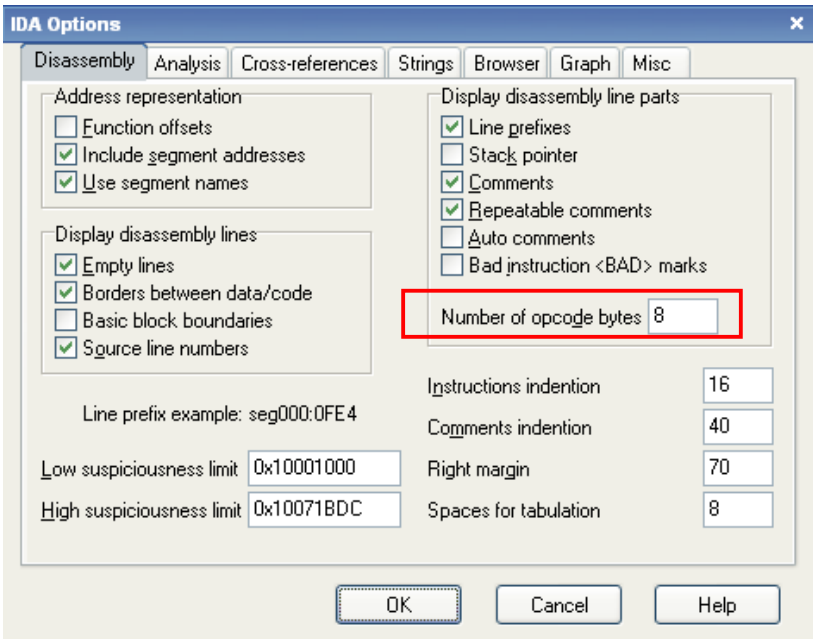
삽질 끝에 언패킹 문제를 해결했다.
이제 dll 파일을 IDA 로 디스어셈블링 해보자.

```
.text:1002ADFB                          ; int __stdcall lib_Decrypt(int,char *)
.text:1002ADFB                          public lib_Decrypt
.text:1002ADFB                          lib_Decrypt     proc near
.text:1002ADFB
.text:1002ADFB                          var_14          = dword ptr -14h
.text:1002ADFB                          var_10          = dword ptr -10h
.text:1002ADFB                          var_C           = dword ptr -0Ch
.text:1002ADFB                          var_8           = dword ptr -8
.text:1002ADFB                          arg_0           = dword ptr  8
.text:1002ADFB                          arg_4           = dword ptr  0Ch
.text:1002ADFB
.text:1002ADFB 55                                       push    ebp
.text:1002ADFC 8B EC                                    mov     ebp, esp
.text:1002ADFE 83 EC 14                                 sub     esp, 14h
.text:1002AE01 E8 87 2C 02 00                           call    sub_1004DA8D
.text:1002AE06 50                                       push    eax
.text:1002AE07 8D 4D F8                                 lea     ecx, [ebp+var_8]
.text:1002AE0A E8 43 29 02 00                           call    MFC42_6467
.text:1002AE0F 8B 45 0C                                 mov     eax, [ebp+arg_4]
.text:1002AE12 50                                       push    eax             ; char *
.text:1002AE13 8B 4D 08                                 mov     ecx, [ebp+arg_0]
.text:1002AE16 51                                       push    ecx             ; int
.text:1002AE17 8D 55 F0                                 lea     edx, [ebp+var_10]
.text:1002AE1A 52                                       push    edx             ; int
```

위의 코드는 INIWebCrypto.dll 파일의 Decrypt 함수가 디스어셈블링된 부분이다. . 여기서 해야 할 일은 Decrypt 함수의 인자값을 후킹으로 가로채서 외장 로거인 DebugView 에 출력할 수 있도록 하는것이 목표이다. 함수로 전달되는 인자값은 int 형과 char 형 포인터이다. 아마도 int 형은 암호화방식의 인덱스 개념일 것이고 char 형 포인터는 복호화할 데이터를 가리키는 주소값일거라 추측해 볼 수 있다.

이제 후킹에 있어 가장 중요한 사항은 가로채려는 지점을 선정하는 기준이다.
그 지점은 ebp를 베이스로 지정할 수 있는 시점부터라면 어디든 상관이 없다. 스택포인터가 제자리를

찾은 후에 코드를 삽입해야 프로그램이 파괴되지 않고 후킹이 진행될 수 있을 것이다.



먼저 opcode를 확인해야 하기 때문에 IDA 설정을 변경해준다.

흐름에 지장이 없으면서 5byte 이상의 코드가 존재하는 곳을 지점으로 삼아야 한다.
Hooking 이 되는 형태 = CALL (1byte) + 상대 address (4byte)

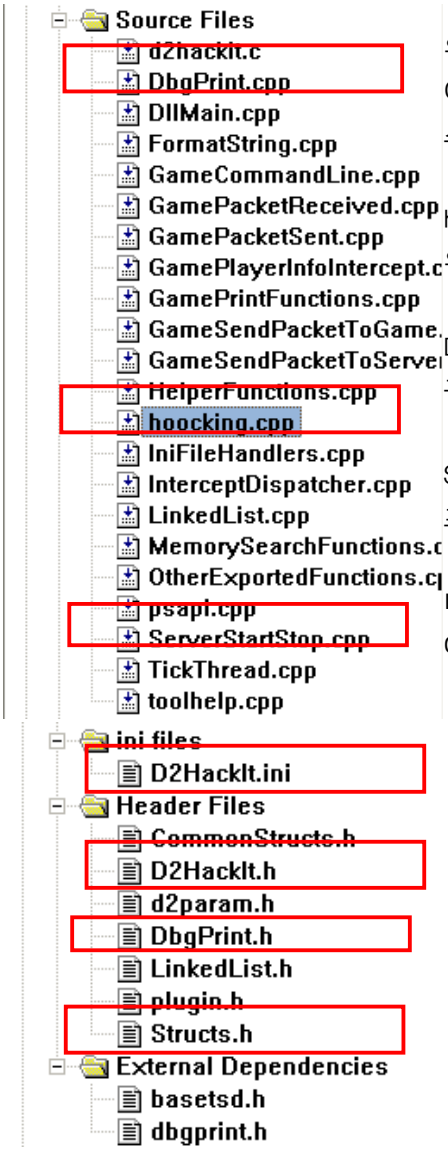
그럼 함수의 시작지점부터 후킹이 가능한 코드를 검색해보자
sub esp,14h 〈- 까지는 스택포인터가 변경되기 때문에 이부분을 건드리게 되면 프로그램이 뻗어버릴 수 있다. 그아래 call 구문도 후킹지점이 될 수 없다(후킹함수 속에서 호출되게 되면 error).
call 아래의 소스들을 확인해보았다.

```
.text:1002AE06 50                                push    eax
.text:1002AE07 8D 4D F8                          lea     ecx, [ebp+var_8]
.text:1002AE0A E8 43 29 02 00                    call    MFC42_6467
.text:1002AE0F 8B 45 0C                          mov     eax, [ebp+arg_4]
.text:1002AE12 50                                push    eax             ; char *
.text:1002AE13 8B 4D 08                          mov     ecx, [ebp+arg_0]
.text:1002AE16 51                                push    ecx             ; int
.text:1002AE17 8D 55 F0                          lea     edx, [ebp+var_10]
.text:1002AE1A 52                                push    edx             ; int
.text:1002AE1B B9 B8 8A 06 10                    mov     ecx, offset unk_10068AB8
.text:1002AE20 E8 0F E7 FD FF                    call    sub_10009534
```

첫 번째 두 번째 두 명령의 OPCODE 의 길이를 합쳐도 5바이트가 되지 못한다.
그밑의 call 구문은 후킹지점으로 부적합하다.
그밑의 push eax부터 push ecx 지점까지 세 개의 명령어를 합치면 5byte가 되지만, 각 OPCODE 는 고정된 명령크기를 갖고있기 때문에 5byte 를 후킹하는건 할 수 없다. 또한 후킹을 하더라도 후킹함수에 3개의 어셈블리 명령이 복사되므로 3byte,1byte,3byte의 총 7byte가 복사되며 후킹함수가 작동할 때마다 실행될 것이다. 3개의 어셈블리 명령이 그 속에서 아무런 탈을 일으키지 않도록 보장하는 코드를 적성하려는 것도 매우 까다로운 작업이다. (push eax 때문에 후킹함수 속에서 4만큼 스택을 증가시키

므로 복구작업을 하지 않으면 리턴 시 엉뚱한 주소로 점프해서 프로그램이 박살나게 됨)
밑에서 두 번째 코드를 보게 되면 OPCODE 도 5byte 이고 ecx 레지스터값 밖에 바꾸지 않아서 스택 복구같은 처리가 따로 필요없다. 이 부분을 후킹하는 코드를 만들면 될 것이다.
이제 후킹을 위해 D2hackit 소스를 수정하여 인젝션할 DLL 파일을 생성하도록 하자.

D2Hackit 소스는 디아블로 채팅창을 후킹하여 해킹을 위한 툴로 제작된 소스이다.
코드들이 모듈화 되어있고, 플러그인 방식으로 제작되었기 때문에 사용자가 원하는 방식대로 수정하여 다른 프로그램에도 적용할수 있는 강력한 후킹 소스이다.



위의 파일중에서 DbgPrint.cpp 와 DbgPrint.h 는 후킹을 수행한 후에 그 정보를 디버깅 창으로 넘기기 위해 새로 제작하여 삽입한 소스이다.

hoocking.cpp 는 후킹지점으로 지정한 코드를 인터셉트하여 후킹을 위해 새로 제작한 함수로 돌리기 위해 제작하여 삽입한 소스이다.

DllMain.cpp 는 Dll 파일의 메인함수라는건 설명안해도 알 것이다. 구조도 간단하기 때문에 특별히 설명하지 않겠다.

ServerStartStop.cpp 는 이 프로그램의 핵심부분으로서 이곳에서 프로그램을 후킹하기 위한 모든 작업을 수행한다.

나머지 빨간박스를 친 부분들 모두 후킹을위해 소스를 수정해 주어야 하는 부분이다.

먼저 DbgPrint.cpp 와 DbgPrint.h 의 소스를 살펴보자

```cpp
#if _MSC_VER > 1000
#pragma once
#endif

#ifndef _debug_
#define _debug_

#include "windows.h"

#ifdef _DEBUG
void DbgPrintf(LPTSTR fmt, ...)
{
    va_list marker;
    TCHAR szBuf[4096];

    va_start(marker, fmt);
    wvsprintf(szBuf,fmt,marker);
    va_end(marker);

    OutputDebugString(szBuf);
    OutputDebugString(TEXT("\r\n"));
}
#endif
#endif
```

〈DbgPrint.cpp〉

```cpp
#if _MSC_VER > 1000
#pragma once
#endif

#ifndef _debug_
#define _debug_

#ifdef _DEBUG
extern void DbgPrintf(LPTSTR fmt, ...);
#endif

#endif
```

〈DbgPrint.h〉

선행처리기를 사용하여 디버그 모드로 컴파일이 될 때 DbgPrintf 라는 디버깅 출력 함수를 생성하는 소스이다. DbgPrintf 함수는 후에 소스에서 디버깅 출력을 위하여 사용된다.

다음은 후킹을 위한 hoocking.cpp 소스이다.

```cpp
#define THIS_IS_SERVER
#include "..\D2HackIt.h"
#include "DbgPrint.h"
/*
Decrypt()
ciphername = 암호화 방식
data = 디코딩 데이터
*/
void __fastcall Decrypt(char *ciphername, char *data)
{
#ifdef _DEBUG
    DbgPrintf("Decrypt = %s, %s\n",ciphername,data);
#endif
}

/*
Decrypt_STUB()
[ebp+0x08] = ciphername
[ebp+0x0c] = data
*/
void __declspec(naked) Decrypt_STUB()
{
    __asm {
        nop
        nop
        nop
        nop
        nop
        nop
        nop

    //.text:1002AE1B B9 B8 8A 06 10   mov     ecx, offset unk_10068AB8
```

먼저 후킹을 위해 Decrypt 함수를 본래의 프로토타입에 맞춰서 새로 선언해 준다. 전달받는 인자값도 동일하게 선언해 준 후에 전달받는 인자값을 디버깅 창으로 출력하는 구문을 작성해준다.

Decrypt_STUB() 함수가 가장 중요한 개념인데 직접 바이너리 코드의 수정을 위해 어셈블리 언어로 수정을 해 준다.
최초에 나오는 nop 코드는 후킹한 지점의(5byte) 본래의 코드가 저장되는 곳이다.

다음에 나오는 어셈코드가 실질적인 후킹작업을 수행하는 코드이다.

```
        pushad
        mov edx, DWORD PTR [ebp+0x0C]
        mov ecx, DWORD PTR [ebp+0x08]
        CALL Decrypt
        popad

        ret
    }
}
```

먼저 후킹을 하면서도 스택영역과 레지스터 영역을 본 상태로 유지하기 위해 pushad 구문으로 모든 레지스트리 정보를 저장한다. 후에 앞에서 새로 생성한 Decrypt 함수의 인자값을 전달하기 위해 본래의 저장되어 있는 위치에서 인자값을 전달하는 레지스터로 값을 복사한 뒤에 Decrypt 함수를 호출하여 함수에 인자값을 전달해 주고, 그 함수에서는 전달받은 인자값을 후킹하여 디버깅 창에 출력해 주는 것이다.

ServerStartStop.cpp 소스는 후킹을 위한 가장 핵심적인 소스이다. 소스에서 디아블로 후킹을 위해 불필요한 부분도 많기 때문에 중요한 부분만 추려서 나타내면 다음과 같다.

```
#define THIS_IS_SERVER
#include "..₩D2HackIt.h"

// These are the dll's we want to force-load to get them in memory.
//char* NeededDlls[] = { "D2Common.dll", "D2Game.dll", "D2Multi.dll", "D2Client.dll", NULL };
//char* NeededDlls[] = { "D2Common.dll", "D2Game.dll", "D2Client.dll", NULL };
char* NeededDlls[] = {  NULL };
// 강제로 로딩하기 위한 DLL 목록을 넣어준다.

BOOL PRIVATE ServerStart(HANDLE hModule)
{
        // Temporary string
        LPSTR t=NULL; // 후킹한 정보들을 담을 구조체들을 초기화한다.(디아블로2를 후킹하는 것이 목적이 아니기에 디아블로 2만의 정보를 담는 구조체는 의미가 없다.


        ////////////////////////////////////////////////////////////////////
        // Before anything else, create the global structures we use in
        // the hack. Make sure we delete these in ServerStop.
        ////////////////////////////////////////////////////////////////////
        si =        new SERVERINFO;
        psi =       new PRIVATESERVERINFO;
        fep =       new FUNCTIONENTRYPOINTS; // fep 구조체는 함수포인터들의 집합을 저장하는 구조체이다.
        pfep=       new PRIVATEFUNCTIONENTRYPOINTS;

        thisgame=new THISGAMESTRUCT;
        thisgame->player=NULL;

        ////////////////////////////////////////////////////////////////////
        // Force-load needed dll's so we can patch thei'r memory space.
        // We should unload these is ServerStop.
        ////////////////////////////////////////////////////////////////////
        // 위의 NeededDlls 에 있는 DLL 들을 여기서 LoadLibrary 로 강제 로딩을 시킨다. 이렇게 함으로서 우리는 타겟 프로그램이 갖고 있는 다른 DLL 들까지도 패치시킬 수 있다.

        for (int i=0; NeededDlls[i] != NULL; i++) {
```

```
		DbgPrintf("LoadDll = %s", NeededDlls[i]);
		LoadLibrary(NeededDlls[i]);
	}

	////////////////////////////////////////////////////////////////
	// Build initial data of the SERVERINFO structure
	////////////////////////////////////////////////////////////////
	si->Version=__SERVERVERSION__; // D2Hackit 의 버전 세팅(의미없음)

	// Get plugin path
	t=new char[_MAX_PATH]; // D2HackIt 은 플러그인 구조를 취하고 있어서 누구나 역공학으로 디아블로2의 핵심 루틴을 찾아서 자신만의 기능을 만들고 플러그인으로 제작할 수 있다. 이 부분이 그 플러그인을 위한 디렉토리의 문자열을 뽑아내는 부분이다.
	if (!GetModuleFileName((HINSTANCE)hModule, t, _MAX_PATH))
	//		{ MessageBox(NULL, "Unable to get PluginPath!", "D2Hackit Error!", MB_ICONERROR); return FALSE; } 밑의 DbgPrint 함수로 대체됨
	{
		#ifdef _DEBUG
		DbgPrintf("Unable to get PluginPath!");
		#endif
		return FALSE;
	}

	//GetModuleFileName 으로 C:₩D2Hackit₩D2HackIt.dll 과 같은 내용이 t 버퍼에 들어왔다면 오른쪽부터 검색해서 첫번째로 역슬래쉬가 나오는 곳까지 자른다.

	int p=strlen(t);
	while (p)
	{
		if (t[p] == '₩₩')
			{ t[p] = 0; p=0;}
		else
			p--;
	}

	si->PluginDirectory=new char[strlen(t)+1];// 이렇게 얻은 플러그인 디렉토리명을 서버정보 구조체 si->PluginDirectory 에 복사한다. (우리가 하려는 작업에 큰 의미는 없음)
	strcpy((LPSTR)si->PluginDirectory, t);

	psi->DontShowErrors=FALSE;
	////////////////////////////////////////////////////////////////
	// Build initial data of the PRIVATESERVERINFO structure
	////////////////////////////////////////////////////////////////
	//앞에서 얻은 디렉토리명과 D2HackIt.init 를 합치면 대략 c:₩D2HackIt₩D2HackIt.ini 가 되며 psi-> iniFile 구조체에 복사한다.
	sprintf(t, "%s₩₩D2HackIt.ini", t);
	psi->IniFile=new char[strlen(t)+1];
	strcpy((LPSTR)psi->IniFile, t);
	delete t;
```

```
        /* This block of code is replaced by the single GetCurrentProcessId()
           call below.  This works because DLL initialization is performed
           within the context of the process to which the DLL is attaching -
           i.e. under that process's PID

        // Get Diablo II's hwnd
        psi->hwnd = FindWindow("Diablo II", "Diablo II");   // Get hwnd
                if (!psi->hwnd) { MessageBox(NULL, "Can't get Diablo II's window handle.", "D2Hackit Error!",
MB_ICONERROR); return FALSE; }

        // Get Diablo II's pid & Process handle


        GetWindowThreadProcessId(psi->hwnd, &psi->pid);
        */

    // Get the process ID and the process handle
        //현재 프로세스 ID 를 얻어서 접근권한을 풀어버린다.
        psi->pid = GetCurrentProcessId();
        psi->hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, psi->pid);
                if (!psi->hProcess) //{ MessageBox(NULL, "Can't get Diablo II's process handle.", "D2Hackit Error!",
MB_ICONERROR); return FALSE;}
                {
                        #ifdef _DEBUG
                        DbgPrintf("Can't Get Module Process handle.");
                        #endif
                        return FALSE;
                }

        // Get build date/time
        //빌드한 시간을 기록하는 것으로 우리 목적에는 별 의미가 없다.
//      strcpy(psi->BuildDate, __DATE__);
//      strcpy(psi->BuildTime, __TIME__);

        ////////////////////////////////////////////////////////////////
        // Build initial callbacks in the FUNCTIONENTRYPOINTS structure.
        ////////////////////////////////////////////////////////////////
// 여기서 주의깊게 봐야할 점은 이름으로도 알 수 있듯이 GamePrint 어쩌구 함수들을 펑션포인터(Function Pointer:함수포인터)
들에 셋팅을 하고 있다는 점이다.


        //      fep->GetMemoryAddressFromPattern=&GetMemoryAddressFromPattern;
//      fep->GamePrintString=&GamePrintString;
//      fep->GamePrintInfo=&GamePrintInfo;
//      fep->GamePrintVerbose=&GamePrintVerbose;
//      fep->GamePrintError=&GamePrintError;
        fep->GetHackProfileString=&GetHackProfileString;

        ////////////////////////////////////////////////////////////////
        // Build initial callbacks in PRIVATEFUNCTIONENTRYPOINTS
        ////////////////////////////////////////////////////////////////
```

```
/* 다음은 프로세스 이미지에 관련된 함수를 셋팅하는 것으로 로딩된 바이너리 이미지의 베이스 주소와 총 크기를 구하기 위해서 작성된
함수의 주소를 구조체에 할당한다. 일단 기본적으로 GetBaseAddress 함수가 베이스이미지 주소를 리턴하고 GetImageSize 함수는 실행파일의 총 크기를 리턴한다는걸 알아두자
*/
	pfep->GetBaseAddress=&GetBaseAddress;
	pfep->GetImageSize=&GetImageSize;

	////////////////////////////////////////////////////////////////////
	// Check if D2Hackli.ini exisits
	////////////////////////////////////////////////////////////////////

	if (_access(psi->IniFile, 0)) // 실제로 D2HackIt.ini 환경설정파일이 있는지 검사한다.
	{
		LPSTR t=new char[strlen(psi->IniFile)+50];
		sprintf(t, "Unable to open ini-file:₩n%s", psi->IniFile);
		#ifdef _DEBUG
		DbgPrintf(t);
		#endif
		delete t;
		return FALSE;
	}
	DbgPrintf("ProcessID = %d₩n",psi->pid);

	//메모리에 로딩된 INIWebCrypto.dll 모듈을 덤프하는 루틴
	HANDLE hFile;
	DWORD dwWritten;

	DWORD BaseAddress;
	DWORD ImageSize;

	//GetBaseAddress 와 GetImageSize 함수는 DbgHelp.dll 의 toolhelp 관련 함수로써 메모리의 프로세스 이미지를 탐색하여 일치하는
	//모듈의 베이스어드레스와 이미지 사이즈를 리턴한다.
	BaseAddress = GetBaseAddress("c:₩₩Program Files₩₩INITECH₩₩INISAFE Web V6₩₩INIWebCrypto.dll");
	ImageSize = GetImageSize("c:₩₩Program Files₩₩INITECH₩₩INISAFE Web V6₩₩INIWebCrypto.dll");

	//INIWebCrypto.dll 파일을 로컬 드라이브 C 에 생성하고 메모리에 로딩된 이미지 내용을 라이팅한다.

hFile=CreateFile("C:₩₩INIWebCrypto.dll",GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
	WriteFile(hFile,(unsigned char *)BaseAddress,ImageSize,&dwWritten,NULL);



	//메모리의 바이너리 이미지(DLL)에서 Decrypt 함수지점 검색
	if (!GetFingerprint("D2HackIt","Decrypt",psi->fps.Decrypt))
	{
		#ifdef _DEBUG
		DbgPrintf("Decrypt Not Found!");
		#endif
/*		// 5초 간격으로 핑거프린팅 쓰레드 생성
```

```
                DWORD dummy =0;
                psi->TickShutDown =0;
                psi->TickThreadHandle=CreateThread(NULL,0,FindTickThread,(void*)0,0,&dummy);
                #ifdef _DEBUG
                DbgPrintf("CreateThread Complete..₩n");
                #endif
                psi->TickThreadActive=psi->TickThreadHandle!=NULL;
*/
                return FALSE;
        }

        //Decrypt 함수부분 인터셉트
        Intercept(INST_CALL,psi->fps.Decrypt.AddressFound,(DWORD)&Decrypt_STUB,psi->fps.Decrypt.PatchSize);
        return TRUE;
        }




        ////////////////////////////////////////////////////////////////////
        // Set default prompts
        ////////////////////////////////////////////////////////////////////
/* 바로 앞에서 GetHackProfileString 이라는 함수를 Fep 구조체로 할당하는 것을 보았을 것이다. GetHackProfileString 함수는
D2Hackit.ini 라는
파일에서 해당하는 섹션을 읽어들여서 값을 뽑아내는 역할을 한다. 다음의 설정은 단순히 디아블로2 채팅창에 출력될 프롬프트를
읽어들이는
것으로 우리는 이 부분도 쓸모가 없다. */
/*      t=fep->GetHackProfileString("D2HackIt", "Misc", "InfoPrompt");
        lstrcpyn(psi->InfoPrompt, (strlen(t)?t:DEFAULTINFOPROMPT),MAXPROMPTLENGTH-1);
        delete t;

        t=fep->GetHackProfileString("D2HackIt", "Misc", "ErrorPrompt");
        lstrcpyn(psi->ErrorPrompt, (strlen(t)?t:DEFAULTERRORPROMPT),MAXPROMPTLENGTH-1);
        delete t;

        t=fep->GetHackProfileString("D2HackIt", "Misc", "VerbosePrompt");
        lstrcpyn(psi->VerbosePrompt, (strlen(t)?t:DEFAULTVERBOSEPROMPT),MAXPROMPTLENGTH-1);

        t=fep->GetHackProfileString("D2HackIt", "Misc", "Verbose");
        if (!stricmp(t, "on"))
                psi->Verbose = TRUE;
        else
                psi->Verbose = FALSE;

        delete t;*/

        ////////////////////////////////////////////////////////////////////
        // Start by binding a way to print to screen. This is vital, so if
        // we are unable to do this, exit with an error message!
        ////////////////////////////////////////////////////////////////////
/* GetFingerprint 함수는 이름으로 알 수 있듯이 핑거프린팅을 하는 함수인데 우리가 환경설정파일에 지정한 바이너리스트링을
```

갖고 메모리에
실행파일이나 DLL 이미지를 대상으로 일치하는 위치(주소)를 찾아내는 역할을 한다. 우리가 마치 울트라에디트 같은 HEX 편집기로 실행파일을
열고 OPCODE 바이너리스트링을 적어서 위치를 찾아내는 것과 같은데 단지 수동이 아니라 지정한 값을 지가 알아서 자동으로 찾는 것이 다르다.
이 함수의 첫번째 인자인 D2HackIt 은 D2HackIt.ini 파일명의 확장자를 뺀 D2HackIt 이고 두번째 인자인 GamePrintStringLocation 문자열은
설정파일의 FingerprintData 섹션에 있는 키 이름을 지정하는 것이다. 만약 GetFingerprint 함수가 메모리에 로딩된 바이너리 이미지로부터
코드주소를 찾아내면 그 위치를 구조체 fps 의 변수인 fps.AddressFound 에 셋팅해서 리턴한다.
*/

```
/*	FINGERPRINTSTRUCT fps;
	if(!GetFingerprint("D2HackIt", "GamePrintStringLocation", fps))
		{ MessageBox(NULL, "Fingerprint information for 'GamePrintStringLocation'₩nmissing or corrupt!", "D2Hackit Error!", MB_ICONERROR); return FALSE; }

	//찾은 주소 fps.AddressFound 를 또 다른 구조체인 psi 의 GamePrintStringLocation 변수로 할당하고 있다.
	psi->GamePrintStringLocation=fps.AddressFound;
	if (!psi->GamePrintStringLocation)
		{ MessageBox(NULL, "Unable to find entrypoint for 'GamePrintStringLocation'!", "D2Hackit Error!", MB_ICONERROR); return FALSE; }


	// Get playerinfo struct
	if (!GetFingerprint("D2HackIt", "pPlayerInfoStruct", fps))
		{ fep->DbgPrintf("Fatal error! Exiting!"); return FALSE; }

	// Messy pointers :)
	thisgame->player=(PLAYERINFOSTRUCT*)*(DWORD*)(*(DWORD*)fps.AddressFound);


	// Get gameinfo struct
	if (!GetFingerprint("D2HackIt", "pPlayerInfoStruct", fps))
		{ fep->DbgPrintf("Fatal error! Exiting!"); return FALSE; }
*/
	// Messy pointers :)
	//thisgame->CurrentGame=(GAMESTRUCT*)*(DWORD*)(*(DWORD*)fps.AddressFound);


	//////////////////////////////////////////////////////////////////
	// Print startup banner.
	//////////////////////////////////////////////////////////////////
```

/* 아래의 GamePrintInfo 함수는 이미 앞에서 보았을 것이다. 이 함수는 디아블로2 바이너리의 채팅창 관련 함수를 강제로 호출하고 있다.
GamePrintString 함수가 결국 GamePrintStringLocation 을 호출하는데 GamePrintInfo 역시 이 출력함수와 연관되어 있다. 그렇기 때문에
fep->GamePrintInfo(t) 명령은 디아블로 2게임의 채팅창에 출력하는 함수이다. 역시 쓰잘데기 없는 내용인데 이렇게 설명하는 이유는
D2HackIt의 기교를 습득하기 위해서 이것 또한 나중에는 이 함수를 다른 함수로 교체할 것이기 때문에 미리 설명하는 것이다.

```
	t=new char[128];
	sprintf(t, "Starting D2HackIt! Mk2 version %d.%.2d (%s@%s)",
		LOWORD(si->Version), HIWORD(si->Version), psi->BuildDate, psi->BuildTime
		);
		fep->GamePrintInfo(t); */

	//////////////////////////////////////////////////////////////////
	// Get loader data
	//////////////////////////////////////////////////////////////////
	// 이 부분은 D2 로더를 사용해서 D2HackIt.dll 을 타겟 프로그램속에 로딩시켯을때 로더의 정보를 얻어내는 부분이다. 기교만 흡수하고 사용하지 않을 것이므로 제거할 것이다.
/*
	BOOL UsingD2Loader=FALSE;
	psi->DontShowErrors=TRUE;
	if (!GetFingerprint("D2HackIt", "LoaderStruct", fps))
	{
		psi->DontShowErrors=FALSE;
		sprintf(fps.ModuleName, "Diablo II.exe");
		UsingD2Loader=TRUE;
		if ((fps.AddressFound=GetMemoryAddressFromPattern(fps.ModuleName, fps.FingerPrint, fps.Offset)) < 0x100)
		{
			sprintf(fps.ModuleName, "D2Loader.exe");
			if ((fps.AddressFound=GetMemoryAddressFromPattern(fps.ModuleName, fps.FingerPrint, fps.Offset)) < 0x100)
			{
				//fep->GamePrintError("Unable to find loader data in 'Game.exe', 'Diablo II.exe' or 'D2Loader.exe'!");
				//fep->GamePrintError("Fatal error! Exiting!"); return FALSE;
				fps.AddressFound=0;
			}
		}
	}

	psi->loader = (LOADERDATA*)fps.AddressFound;

	if (psi->loader)
	{
		sprintf(t, "Loader version c4%d.%.2d c0Game is %sstarted with D2Loader.",
			LOWORD(psi->loader->LoaderVersion), HIWORD(psi->loader->LoaderVersion),
			(UsingD2Loader?"":"c4not c0"));
	} else {
		sprintf(t, "D2Hackit was loaded without loader");
	}
	fep->GamePrintInfo(t);
*/

	//////////////////////////////////////////////////////////////////
	// Continue binding entrypoints and intercepts
	//////////////////////////////////////////////////////////////////
	/*채팅창에 GamePrintStringLocation 을 핑거프린팅으로 찾았다는 메시지 출력.. 제거하거나 교체한다.
```

```
		sprintf(t, "Found 'GamePrintStringLocation' at %.8x", psi->GamePrintStringLocation);
		fep->DbgPrintf(t); */

		/*
		 * No need for this
		 *
		// Get socket location
		if (!GetFingerprint("D2HackIt", "pGameSocketLocation", fps))
			{ fep->GamePrintError("Fatal error! Exiting!"); return FALSE; }
		psi->GameSocketLocation=*(DWORD*)fps.AddressFound;
		*/

		// Get GamePacketReceivedIntercept


/*지금부터 이 ServerStart 함수 내에서 가장 중요한 부분이라고 할 수 있다.
설정파일에서 GamePacketReceivedIntercept 에 해당하는 바이너리스트링을 읽어들여 핑거프린트 함수로 주소를 찾으면
psi->fps.GamePacketReceivedIntercept 변수에 저장한다.
		if (!GetFingerprint("D2HackIt", "GamePacketReceivedIntercept", psi->fps.GamePacketReceivedIntercept))
			{ fep->DbgPrintf("Fatal error! Exiting!"); return FALSE; } */

/* 위에서 핑거프린트로 디아블로 2의 패킷수신부 주소를 찾으면 해당 주소지점을 Intercept 함수로 가로채어
GamePacketReceivedInterceptSTUB
함수로 라우팅 시키는 코드이다. 실제적인 작동은 INST_CALL 에 해당하는 0xE8 1byte와 4byte
GamePacketReceivedInterceptSTUB 함수주소를 합친
총 5byte 크기의 명령을 AddressFound 주소지점에 덮어쓴다. 이때 덮어지기 전의 5byte 명령을
GamePacketReceivedInterceptSTUB 함수의 앞부분에
마련한 NOP 으로 채워진 빈 공간에 복사한다. 이때 PatchSize 는 덮어지는 위치(주소지점)의 명령어가 몇 바이트 명령어인지 지
정한다.

		Intercept(INST_CALL, psi->fps.GamePacketReceivedIntercept.AddressFound,
(DWORD)&GamePacketReceivedInterceptSTUB, psi->fps.GamePacketReceivedIntercept.PatchSize);

		// Get GamePacketSentIntercept
		if (!GetFingerprint("D2HackIt", "GamePacketSentIntercept", psi->fps.GamePacketSentIntercept)) //패킷송신부 핑거
프린팅
		{
			fep->DbgPrintf("Fatal error! Exiting!");
			//만약 패킷송신부를 찾다 실패하면 이전에 인터셉트한 부분을 다시 원상복귀 시켜줘야 하므로 위와 다르게
Intercept 함수에 넘겨지는 인자가 반대로 되어있다.
			Intercept(INST_CALL, (DWORD)&GamePacketReceivedInterceptSTUB,
psi->fps.GamePacketReceivedIntercept.AddressFound, psi->fps.GamePacketReceivedIntercept.PatchSize);
			return FALSE;
		}
		//위와 마찬가지로 패킷송신부를 인터셉트한다.
		Intercept(INST_CALL, psi->fps.GamePacketSentIntercept.AddressFound,
(DWORD)&GamePacketSentInterceptSTUB, psi->fps.GamePacketSentIntercept.PatchSize);
*/

		// Get GamePlayerInfoIntercept
		// 이하 GetFingerPrint 와 Intercept 들도 마찬가지의 작업이다. 분석할 필요는 없다. 뒤에서 이 부분을 모두 삭제할
```

것이다.

```
/*		if (!GetFingerprint("D2HackIt", "GamePlayerInfoIntercept", psi->fps.GamePlayerInfoIntercept))
		{
			fep->GamePrintError("Fatal error! Exiting!");
			Intercept(INST_CALL, (DWORD)&GamePacketSentInterceptSTUB,
psi->fps.GamePacketSentIntercept.AddressFound, psi->fps.GamePacketSentIntercept.PatchSize);
			Intercept(INST_CALL, (DWORD)&GamePacketReceivedInterceptSTUB,
psi->fps.GamePacketReceivedIntercept.AddressFound, psi->fps.GamePacketReceivedIntercept.PatchSize);
			return FALSE;
		}
		Intercept(INST_CALL, psi->fps.GamePlayerInfoIntercept.AddressFound, (DWORD)&GamePlayerInfoInterceptSTUB,
psi->fps.GamePlayerInfoIntercept.PatchSize);


		// Get GameSendPacketToGameLocation
		// Thanks to TechWarrior
		if (!GetFingerprint("D2HackIt", "GameSendPacketToGameLocation", fps))
		{
			fep->GamePrintError("Fatal error! Exiting!");
			Intercept(INST_CALL, (DWORD)&GamePacketSentInterceptSTUB,
psi->fps.GamePacketSentIntercept.AddressFound, psi->fps.GamePacketSentIntercept.PatchSize);
			Intercept(INST_CALL, (DWORD)&GamePacketReceivedInterceptSTUB,
psi->fps.GamePacketReceivedIntercept.AddressFound, psi->fps.GamePacketReceivedIntercept.PatchSize);
			Intercept(INST_CALL, (DWORD)&GamePlayerInfoInterceptSTUB,
psi->fps.GamePlayerInfoIntercept.AddressFound, psi->fps.GamePlayerInfoIntercept.PatchSize);
			return FALSE;
		}
		psi->GameSendPacketToGameLocation=fps.AddressFound;


		// Start TickThread, We dont care about closing it later.
		// It will be destroyed when unloading the dll.
		// 잘은 모르겠으나 클라이언트 정보를 주기적으로 뭔가 하려는 목적으로 쓰레드를 돌리는 것으로 파악된다. (필요치
않으므로 제거)
		DWORD dummy=0;
		psi->TickThreadHandle = CreateThread(NULL,0,TickThread,(void*)&ClientList,THREAD_TERMINATE,&dummy);
		psi->TickThreadActive=TRUE;
*/
		// Load any clients listed in Autorun
		//이하 디아블로2에 관련된 것이므로 필요없으므로 제거할 내용이다.
/*		t=new char[1024];
		t=fep->GetHackProfileString("D2HackIt", "Misc", "Autoload");

		if (strlen(t))
		{
			char* command[2];
			command[0]=".load";
			char *p;
			p=t;

			command[1] = p;
```

```
                while (*p != 0) {
                        if (*p == ',') {
                                *(p++) = 0;
                                GameCommandLineLoad(command,2);
                        while (*p == ' ')
                                                p++;
                                if (*p != 0)
                                        command[1] = p;
                                } else
                                p++;
                        }
                        GameCommandLineLoad(command,2);
        }

        delete t;

        fep->GamePrintInfo("D2HackIt! Mk2 Loaded! Type c4.help0 for help on commands.");
        return TRUE;
}
*/


////////////////////////////////////////////////////////////////////////
// ServerStop()
// ----------------------------------------------------------------------
// Responsible for stopping the server.
////////////////////////////////////////////////////////////////////////
// 다음은 프로그램이 종료하면서 DLL 이 DETACH(언로드) 될 때 실행되는 ServerStop 함수이다.

BOOL PRIVATE ServerStop(void)
{
        // Kill our tickthread
        // 위에서 생성한 틱(Tick) 쓰레드를 죽이는 짓인데 우리는 쓰레드를 만들지 않을것이므로 죽이지도 않을 것이다.
        //TerminateThread(psi->TickThreadHandle, 1234);
        //CloseHandle(psi->TickThreadHandle);

        // Unload any loaded clients
        //클라이언트 정보를 해제하는데 우리는 이 부분이 필요치 않다. 디아블로2를 분석해야 뭔짓을 하는지 알 것 같다.
/*      while (ClientList.GetItemCount())
        {
                LinkedItem *li=ClientList.GetLastItem();
                CLIENTINFOSTRUCT *cds=(CLIENTINFOSTRUCT*)li->lpData;
                char t[32];
                sprintf(t, "unload %s", cds->Name);
                GameCommandLine(t);
        }
*/

        // Un-patch intercept locations
        // 이 부분이  ServerStop 함수에서 가장 중요하다. 프로그램이 시작되고 Intercept 가 되었을 것이므로 다시 복구해
줘야 한다
```

```
        //Intercept(INST_CALL, (DWORD)&GamePlayerInfoInterceptSTUB, psi->fps.GamePlayerInfoIntercept.AddressFound, psi->fps.GamePlayerInfoIntercept.PatchSize);
        //Intercept(INST_CALL, (DWORD)&GamePacketSentInterceptSTUB, psi->fps.GamePacketSentIntercept.AddressFound, psi->fps.GamePacketSentIntercept.PatchSize);
        //Intercept(INST_CALL, (DWORD)&GamePacketReceivedInterceptSTUB, psi->fps.GamePacketReceivedIntercept.AddressFound, psi->fps.GamePacketReceivedIntercept.PatchSize);

        // Release dll's that we loaded upon entry.
        // 강제로 로딩했던 DLL 들을 해제시켜준다.
        for (int i=0; NeededDlls[i] != NULL; i++) FreeLibrary (GetModuleHandle(NeededDlls[i]));

        //fep->GamePrintInfo("D2HackIt! Mk2 Unloaded."); // 언로드 되었다고 디아블로2채팅창에 출력

        delete (LPSTR)si->PluginDirectory, (LPSTR)psi->IniFile, thisgame, pfep, fep, psi, si; // 메모리를 할당한 변수들을 제거

        return TRUE;
}
```

소스가 길어서 읽기가 불편한데 후킹을위한 핵심적인 부분은 다음과 같다.

```
if (_access(psi->IniFile, 0)) // 실제로 D2HackIt.ini 환경설정파일이 있는지 검사한다.
{
    LPSTR t=new char[strlen(psi->IniFile)+50];
    sprintf(t, "Unable to open ini-file:\n%s", psi->IniFile);
    #ifdef _DEBUG
    DbgPrintf(t);
    #endif
    delete t;
    return FALSE;
}
DbgPrintf("ProcessID = %d\n",psi->pid);

//메모리에 로딩된 INIWebCrypto.dll 모듈을 덤프하는 루틴
HANDLE hFile;
DWORD dwWritten;

DWORD BaseAddress;
DWORD ImageSize;

//GetBaseAddress 와 GetImageSize 함수는 DbgHelp.dll 의 toolhelp 관련 함수로써 메모리의 프로세스 이미지를
//모듈의 베이스어드레스와 이미지 사이즈를 리턴한다.
BaseAddress = GetBaseAddress("c:\\Program Files\\INITECH\\INISAFE Web V6\\INIWebCrypto.dll");
ImageSize = GetImageSize("c:\\Program Files\\INITECH\\INISAFE Web V6\\INIWebCrypto.dll");

//INIWebCrypto.dll 파일을 로컬 드라이브 C 에 생성하고 메모리에 로딩된 이미지 내용을 라이팅한다.
hFile=CreateFile("C:\\INIWebCrypto.dll",GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,NULL);
WriteFile(hFile,(unsigned char *)BaseAddress,ImageSize,&dwWritten,NULL);
```

위의 소스는 후킹에 관여하는 소스는 아니다. 단지 후킹작업도중에 메모리에 로딩된 dll 과 파일 상에서 존재하는 dll 의 차이점을 분석하기 위하여 메모리에 로딩된 이미지를 덤프하는 작업이다.
후킹을 위하여 D2Hackit.ini 파일에서 바이너리 스트링 값을 이용해 후킹작업을 하기 위해 지정한 코드의 위치를 찾아가게 된다. 이때 파일상에서의 바이너리 값과 메모리 상에서의 바이너리 값이 차이가 발생할 수 있는데 이는 PE파일의 구조와 관련이 있다. (PE파일의 구조와 언패킹의 원리 참조)
여기서 발생하는 파일 상에서의 바이너리스트링 값과 메모리 상에서의 바이너리스트링을 비교하여 변경되는 부분을 다음과 같이 처리해 주면 D2HackIt 툴이 함수를 검색하여 후킹을 수행할 수 있다.

```
[FingerprintData]
; Decrypt
Decrypt=c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll,5,12,8B450C508B4D08518D55F052xxxxxxxxxxE80FE7FDFF8D4DF0
```

〈D2Hackit.INI 파일의 내용〉

위에서 xxxxx... 이 부분이 코드가 변경되는 부분이다. 와일드 문자라고 해석하면 되겠다.

컴파일후에 생성된 D2HackIt.dll 은 다른 PE 파일에 삽입이 된 후에 동일한 디렉토리에서 D2HackIt.ini 파일의 내용을 참조하여 후킹을 수행하게 된다. 위에 문장을 해석해보면 Decrypt 함수를 후킹하는 것인데 함수가 저장된 모듈(DLL)의 경로와 후킹할 byte(5) 바이너리스트링으로부터의 offset(12) 후킹지점을 식별하기 위한 바이너리스트링으로 구성된 문장이다.

serverstartstop.cpp 의 소스를 계속 살펴보면 다음과 같다.

```
    //메모리의 바이너리 이미지(DLL)에서 Decrypt 함수지점 검색
    if (!GetFingerprint("D2HackIt","Decrypt",psi->fps.Decrypt))
    {
        #ifdef _DEBUG
        DbgPrintf("Decrypt Not Found!");
        #endif
/*      // 5초 간격으로 펑거프린팅 쓰레드 생성
        DWORD dummy =0;
        psi->TickShutDown =0;
        psi->TickThreadHandle=CreateThread(NULL,0,FindTickThread,(void*)0,0,&dummy);
        #ifdef _DEBUG
        DbgPrintf("CreateThread Complete..₩n");
        #endif
        psi->TickThreadActive=psi->TickThreadHandle!=NULL;
*/
        return FALSE;
    }

    //Decrypt 함수부분 인터셉트
    Intercept(INST_CALL,psi->fps.Decrypt.AddressFound,(DWORD)&Decrypt_STUB,psi->fps.Decrypt.PatchSize);
    return TRUE;
    }
```

가장 중요한 부분인 GetFingerprint 함수와 Intercept 함수가 나오는 부분이다.

GetFingerPrint 함수는 메모리의 바이너리 이미지에서 지정한 함수를 검색하여 함수의 위치를 찾아내는 함수이다. 여기서 검색하는 함수는 당연히 후킹을 위한 함수일 것이다.

주석처리된 쓰레드 생성 부분은 처음에 컴파일후에 계속 시행착오를 겪게되면서 시도했던 부분 중 하나인데 처음에 문서에 나온대로 INIWebCrypto.dll 에도 후킹 DLL 을 삽입해보고, iexplore.exe에도 DLL 을 삽입하여 후킹을 계속 시도하였지만 아무리 해도 Decrypt 함수를 찾지 못하였다. 그래서 DLL 이 로딩되는 시간차 때문인지 의심스러워 쓰레드를 생성하여 주기적으로 Decrypt 함수의 검색을 시도한 흔적이다.

아래에 Intercept 함수에서 찾아낸 함수의 주소를 실질적으로 후킹작업을 수행하는 부분이다.

이외에도 수정할 부분이 많지만 Art of Hoocking 문서를 참고하도록 한다.

소스가 수정이완료되고 컴파일되면 D2Hackit.dll 파일이 생성이 된다. 이 파일을 메모리에 상주시키기 위해 어떤 파일에 삽입을 할까 고민해보아야 하는데 최적은 후킹할 지점이 포함된 INIWebcrypto.dll 안에 끼워넣으면 될 것이다. 그러나 무슨 이유인지 INIWebcrypto.dll 파일 안에 삽입하게되면 익스플로어에서 보안모듈이 호출이 될 때 호출할 수 없다는 에러가 발생하였다. 그렇다면 익스플로어가 실행될때 같이 호출되어 미리 후킹을 수행할 DLL 들도 강제로 호출한 후에 후킹작업을 해 놓으면 보안모듈이 실행될때 코드의 흐름은 후킹된 쪽으로 흘러갈 것이다.

익스플로어에 후킹 모듈을 삽입하였으나 강제로 로딩한 WebCrypto.dll 과 INISAFEWeb60.dll 이 강제

로 로딩되지가 않았다. 때문에 디버깅 창에서는 Decrypt 함수를 찾을 수 없다는 메시지만 반복될 뿐이었다.

온갖 방법과 시행착오를 겪고 난 후..

system32 디렉토리에 따로 저장되어 있는 inicrypto30.dll 모듈에 후킹 모듈을 삽입하였다.

```
C:₩WINDOWS₩system32>setdll /d:D2HackIt.dll inicrypto30.dll
Adding D2HackIt.dll to binary files.
  inicrypto30.dll:
    D2HackIt.dll
    WSOCK32.dll -> WSOCK32.dll
    GDI32.dll -> GDI32.dll
    ADVAPI32.dll -> ADVAPI32.dll
    USER32.dll -> USER32.dll
    MSVCRT.dll -> MSVCRT.dll
    KERNEL32.dll -> KERNEL32.dll
```

다시 보안 모듈이 실행되는 사이트로 접속하여 후킹이 진행되는지 확인해 보았다...결과는

```
6614 1115.84741211  [1164]
6615 1115.84753418  [1164] lpme->szModule = ddrawex.dll, Module = INIWebCrypto.dll, ModuleName =c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll
6616 1115.84753418  [1164]
6617 1115.84765625  [1164] lpme->szModule = DDRAW.dll, Module = INIWebCrypto.dll, ModuleName =c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll
6618 1115.84765625  [1164]
6619 1115.84765625  [1164] lpme->szModule = DCIMAN32.dll, Module = INIWebCrypto.dll, ModuleName =c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll
6620 1115.84765625  [1164]
6621 1115.84765625  [1164] lpme->szModule = MSOXMLMF.DLL, Module = INIWebCrypto.dll, ModuleName =c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll
6622 1115.84777832  [1164]
6623 1115.84777832  [1164] lpme->szModule = INIWebCrypto.dll, Module = INIWebCrypto.dll, ModuleName =c:₩Program Files₩INITECH₩INISAFE Web V6₩INIWebCrypto.dll
6624 1115.84777832  [1164]
6625 1115.85021973  [1164] Found 'Decrypt' at 0620ae1b
6626 1115.85021973  [1164]
6627 1115.85021973  [1164] Code at 0620ae1b intercepted and routed to 03eb4233
6628 1115.85021973  [1164]
```

메모리에 로드된 모듈들을 검색하여 INIWebCrypto.dll 을 찾아내었고 그곳에서 Decrypt 함수를 후킹할 수 있었다.