The About Technique of Anti-Debugging

Writeen by hubeen

안티 디버깅(Anti-Debugging) : 디버깅을 방지하고 분석을 하지 못하도록 하는 기술

이 안티 디버깅을 적용함으로써 디버깅을 방지하고 분석을 방해하며 현재 디버깅을 당하고 있다면 디버깅을 하지 못하도록 프로그램을 종료시키거나 에러를 발생 시키는 방법 등 다양한 방법을 사용하여 분석을 방해한다.

❖ **Anti-Debugging 기법 간단 소개**

Technique of Anti-Debugging	**Explanation**
IsDebuggerPresent	PEB 구조체의 디버깅 상태값을 확인하여 디버깅을 당하고 있다면 1, 아닐 경우에는 0 을 리턴합니다.
IsDebugged	PEB 구조체의 BeingDebugged 멤버의 값을 확인하여 디버깅을 당하고 있는지의 여부를 알려준다.
NtGlobalFlag	PEB 구조체에서 0x68 위치에 있는 NtGlobalFlag 의 값을 확인하여 체크합니다 보통 디버깅을 당했을 경우 0x70, 아닐 경우에는 0 이 리턴됩니다.
CheckRemoteDebuggerPresent	Windows XP 이상부터 사용할 수 있으며 ZwQueryInformationProcess 를 사용하여 프로세스의 DebugProt 정보를 얻게 되는데 0 이면 정상, 디버깅을 당했을 경우 다른 값을 리턴한다.
FindWindow	FindWindow API 를 사용하여 특정 윈도우 이름이나 클래스 이름을 찾아 특정 프로그램이 실행중인지를 확인합니다.

위의 표의 설명에 자주 보이는 구조체에 대해 설명을 하도록 하겠습니다.
보통 현재 프로세스 디버깅을 판단하기 위해 PEB 구조체 정보를 이용합니다.
PEB 구조체 정보를 이용하는 것이 현재 가장 잘 사용되는 안티 디버깅 기법입니다.
PEB 구조체 주소는 FS Segment Register 가 가르키는 TEB 구조체를 이용하여 구할 수 있습니다.

Struct	**Explanation**
TEB	프로세스에서 실행되는 스레드에 대한 정보를 담고 있는 구조체입니다.
PEB	프로세스의 정보를 담고 있는 구조체입니다.

아래는 PEB 와 TEB 의 구조체입니다.

```
typedef struct _PEB {
        BOOLEAN InheritedAddressSpace;
        BOOLEAN ReadImageFileExecOptions;
        BOOLEAN BeingDebugged;
        BOOLEAN Spare;
         ...
         ULONG PostProcessInitRoutine;
         ULONG TlsExpansionBitmap;
         BYTE TlsExpansionBitmapBits[0x80];
         ULONG SessionId;
} PEB *PPEB;

Typedef struct _TEB {
        BYTE Reserved1[1952];
        PVOID Reserved2[412];
        PVOID TlsSlots[64];
        BYTE Reserved3[8];
        PVOID Reserved4[26];
        PVOID ReservedForOle;
        PVOID Reserved5[4];
        PVOID  TlsExpansionSlots;
} TEB, *PTEB;
```

이러한 기법들은 OS 의존성이 있기 때문에 사전에 이를 확인하는 것이 좋습니다.
이러한 기법들을 적용시키는 디버거 플러그인들도 있으며, 계속 변형된 버전이 나오고 있습니다.
이러한 기법들을 간단히 우회할 수 있는 플러그인들이 있지만,
계속 변형된 버전이 나오기 때문에 동작 원리와 기본 회피 방법들에 대해 공부를 하는 것이 큰 도움이 됩니다.

아래에서는 위에서 소개한 기법들에 대한 회피 방법을 작성하겠습니다.

❖ Bypass Anti-Debugging

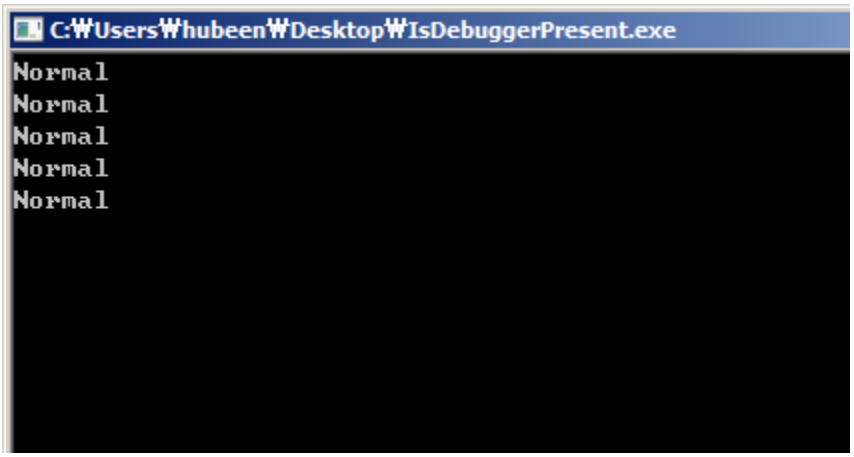
1. IsDebuggerPresent()

이 API 는 PEB 구조체에 있는 BeingDebugged 값을 참조하여 디버깅 여부를 판변합니다.
이 API 는 kernel32.dll 에서 export 되는 함수이다.
BeingDebugged 값은 PEB 구조체 + 0x02 에 위치해 있습니다.

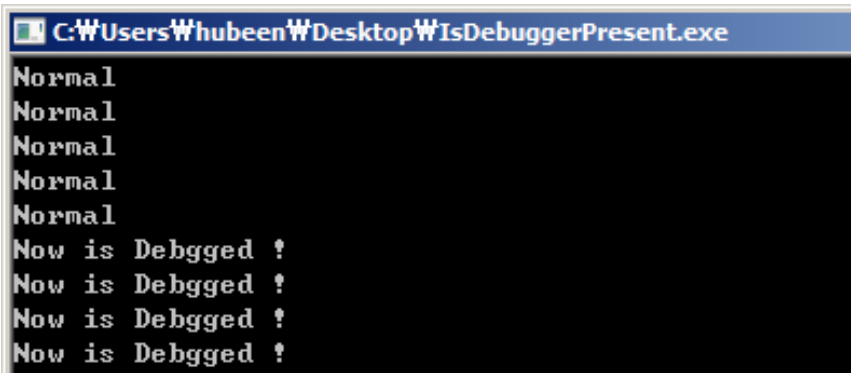
아래는 IsDebuggerPresent()의 샘플 코드입니다.

```
#define _WIN32_WINNT 0X0500
#include <stdio.h>
#include <Windows.h>

int main()
{
	while (TRUE)
	{
		Sleep(1000);
		if (IsDebuggerPresent())
		{
			printf("Now is Debgged !₩n");
		}
		else
		{
			printf("Normal₩n");
		}
	}
}
```

위의 코드를 디버깅하여 실행을 해보면 아래와 같은 메세지를 출력합니다.



현재 디버깅을 당하지 않았기 때문에 정상이라는 메세지를 출력하게 됩니다.
이제 이 프로그램을 Attach 를 하게 되면 아래와 같이 메세지를 출력하게 됩니다.



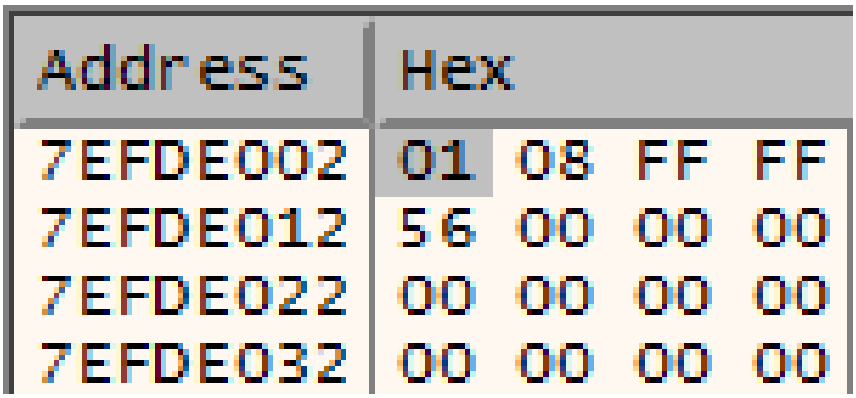
이제 디버깅 툴로 IsDebuggerPresent 를 따라가보도록 하겠습니다.

```
call dword ptr ds:[<&IsDebuggerPresent>]
cmp esi,esp
call isdebuggerpresent.F1113
test eax,eax
je isdebuggerpresent.F17DD
push isdebuggerpresent.F6B30          000F6B30:"Now is Debgged !\n"
call isdebuggerpresent.F1320
add esp,4
jmp isdebuggerpresent.F17EA
push isdebuggerpresent.F6B48          000F6B48:"Normal\n"
call isdebuggerpresent.F1320
jmp <kernel32.IsDebuggerPresent>
nop
nop
nop
nop
nop
```

위의 ds:[<&IsDebuggerPresent>]를 들어가보면 아래의 이미지에서 볼 수 있듯이
커널 영역에서 호출됨을 확인할 수 있었습니다.

```
mov eax,dword ptr fs:[18]
mov eax,dword ptr ds:[eax+30]
movzx eax,byte ptr ds:[eax+2]
ret
```

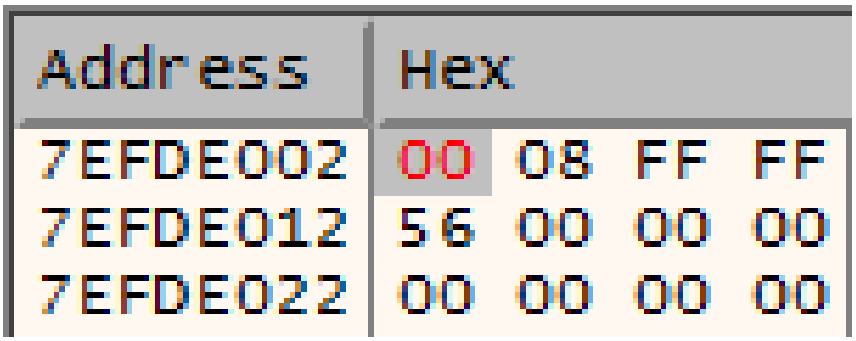
계속 따라가다보면 TEB 구조체를 이용하여 PEB 구조체를 찾는 모습을 볼 수 있습니다.
TEB 주소 : FS[0x18]
PEB 주소 : DS[TEB 주소+0x30]
그리고 PEB 에서 0x2 한 것이 BeingDebugged 값이 되게 됩니다.



Address	Hex
7EFDE002	01 08 FF FF
7EFDE012	56 00 00 00
7EFDE022	00 00 00 00
7EFDE032	00 00 00 00

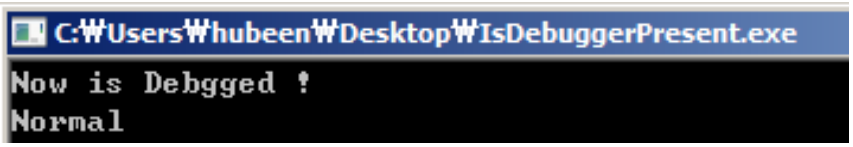
PEB 에서 0x2 에 1 이 들어간 것을 볼 수 있습니다.

그러므로 이 eax 값을 0 으로 패치하면 자연스럽게 우회를 할 수 있게 됩니다.



Address	Hex
7EFDE002	00 08 FF FF
7EFDE012	56 00 00 00
7EFDE022	00 00 00 00

이렇게 값을 0 으로 패치를 해주게 되면.



간단히 우회가 된 것을 볼 수 있습니다.

2. NtGlobalFlag

이 기법은 NtGlobalFlag 값을 이용하여 디버깅 여부를 판단하는 방법입니다.
PEB 의 0x68 에 위치에 있는 NtGlobalFlag 값이 0 이면 정상 아니면 디버깅을 당한 것으로 볼 수 있습니다.

대개 디버깅을 당하게 되면 0x70 이 설정되는데

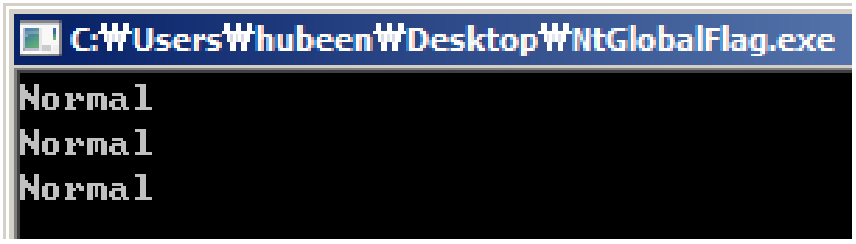
아래의 FLAG 값들이 더해진 결과입니다.

FLG_HEAP_ENABLE_TAIL_CHECK(Heap Tail Checking)	0x10
FLG_HEAP_ENABLE_FREE_CHECK(Heap Free Checking)	0x20
FLG_HEAP_VALIDATE_PARAMETERS(Heap Parameter Checking)	0x40

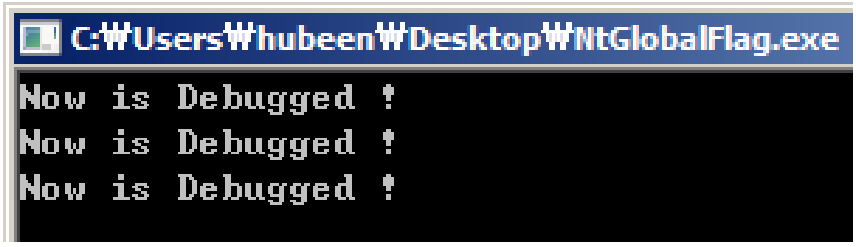
아래는 NtGlobalFlag 의 샘플 코드입니다.

```
#define _WIN32_WINT 0x0500
#include <stdio.h>
#include <Windows.h>

int main()
{
	while (TRUE)
	{
		Sleep(1000);
		if (NtGlobalCheck()!=0)
		{
			printf("Now is Debugged !₩n");
		}
		else
		{
			printf("Normal₩n");
		}
	}
}

int NtGlobalCheck()
{
	_asm
	{
		mov eax, fs:[30h]
		mov eax, [eax+68h]
		and eax, 0x70
		test eax, eax
	}
}
```

위의 코드를 디버깅한 뒤에 실행을 해보면 아래와 같은 메세지를 출력합니다.

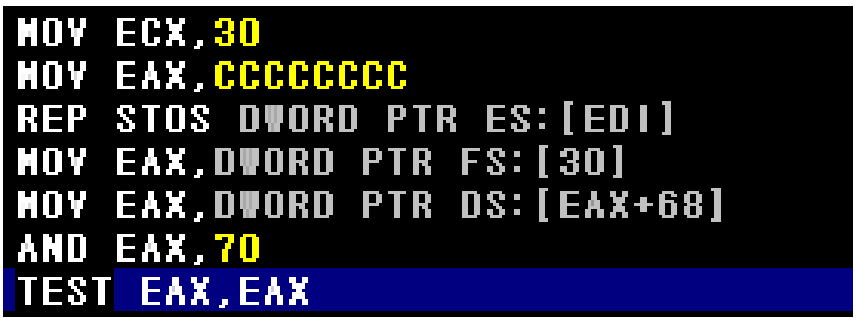


현재 디버깅을 당하지 않았기 때문에 정상이라는 메세지를 출력하게 됩니다.
이제 이 프로그램을 디버깅 툴에서 실행하게 되면 아래와 같이 메세지를 출력하게 됩니다.

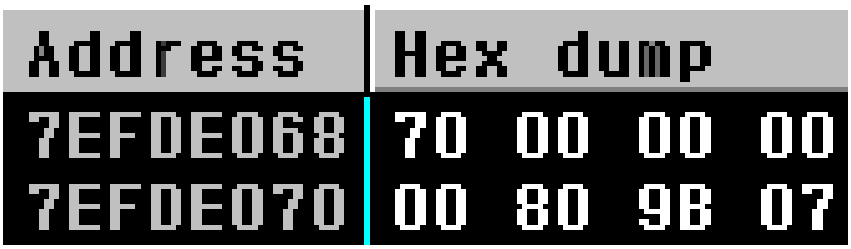


이제 디버깅 툴로 따라가보도록 하겠습니다.

위에 있는 함수를 들어가게 되면 아래와 같은 어셈 코드를 발견할 수 있습니다.

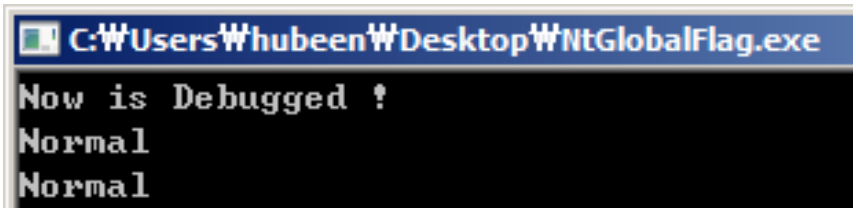


이 어셈 코드는 0x70 과 비교를 하여 디버깅 여부를 확인하는 코드입니다.
PEB 의 주소 값 +0x68 에 NtGlobalFlag 값이 있음을 알고 있으므로
저 주소에 담겨 있는 값을 0 으로 패치를 해주게 되면 우회가 될 수 있음을 알 수 있습니다.



70 이 고스란히 들어있음을 볼 수 있습니다.

0 으로 패치를 해줍니다.



간단히 우회가 됐음을 볼 수 있습니다.

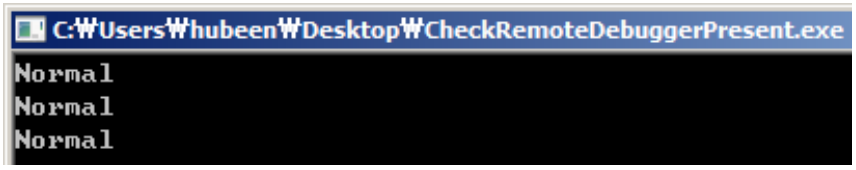
3. CheckRemoteDebuggerPresent()

이 API 는 windows XP 이상부터 사용을 할 수 있으며
내부적으로 ZwQueryInformationProcess API 를 이용하여
프로세스의 DebugProt 정보를 통해 디버깅 당함을 체크합니다.

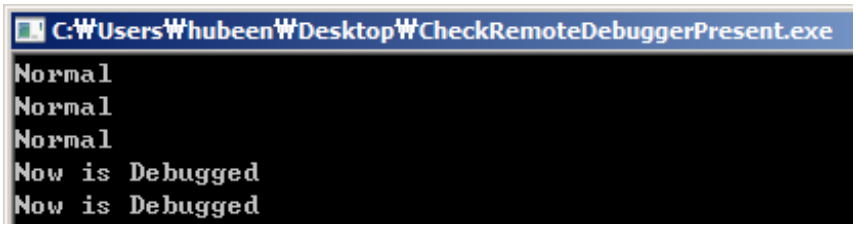
아래는 CheckRemoteDebuggerPresent 의 샘플 코드입니다.

```
#define _WIN32_WINNT 0x0501
#include <stdio.h>
#include <windows.h>

int main()
{
	BOOL bDebugged = FALSE;
	while (TRUE)
	{
		Sleep(1000);
		CheckRemoteDebuggerPresent(GetCurrentProcess(), &bDebugged);
		if (bDebugged)
			printf("Now is Debugged₩n");
		else
			printf("Normal₩n");
	}
	return 0;
}
```

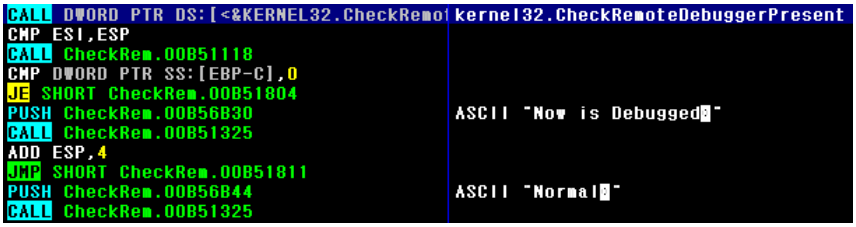
위의 코드를 디버깅한 뒤에 실행을 해보면 아래와 같은 메세지를 출력합니다.



현재 디버깅을 당하지 않았기 때문에 정상이라는 메세지를 출력하게 됩니다.
이제 이 프로그램을 Attach 하게 되면 아래와 같이 메세지를 출력하게 됩니다.



이제 디버깅 툴로 따라가보도록 하겠습니다.



Ebp-c 와 0 을 비교하는 것을 발견할 수 있습니다.

우회 방법은 그 밑에 점프문을 통해 우회를 시켜주면 됩니다.
또는 인자 값을 변경해주는 방법도 있습니다.

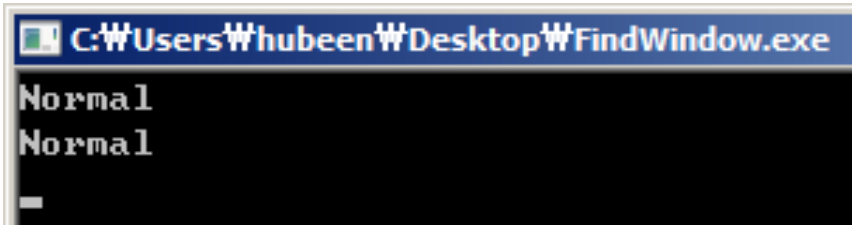
4. FindWindow

이 함수는 잘 알려져있는 이름을 검색하여 그 조건이 맞다면 디버깅을 중지시키는 기법입니다.

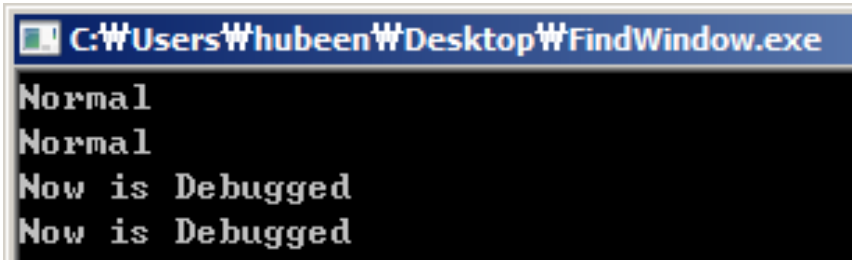
아래는 FindWindow 샘플 코드입니다.

```
#define _WIN32_WINNT 0x500
#include <stdio.h>
#include <windows.h>

int Check();

int main()
{
	while (TRUE)
	{
		Sleep(1000);
		if (Check() != 0)
			printf("Now is Debugged₩n");
		else
			printf("Normal₩n");
	}
	return 0;
}

int Check()
{
	char debugger[] = "OLLYDBG";
	int ret;
	_asm
	{
		push 0		// WindowName
		lea eax, debugger
		push eax	// ClassName
		call dword ptr FindWindowA
		mov ret, eax
	}
	return ret;
}
```

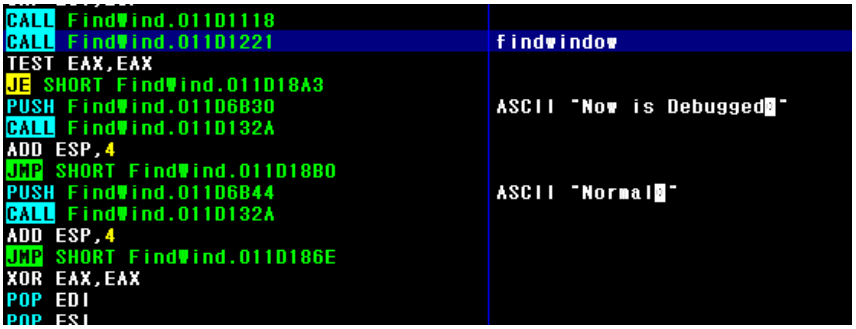
위의 코드를 디버깅한 뒤에 실행을 해보면 아래와 같은 메세지를 출력합니다.



현재 올리디버거를 키지 않았기 때문에 정상이라는 메세지를 출력하게 됩니다.
이제 올리디버거를 킨 뒤에 메세지를 확인해보면 아래와 같이 메세지를 출력하게 됩니다.

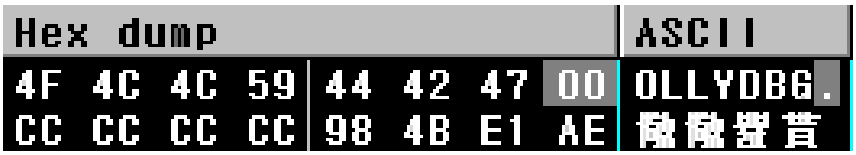


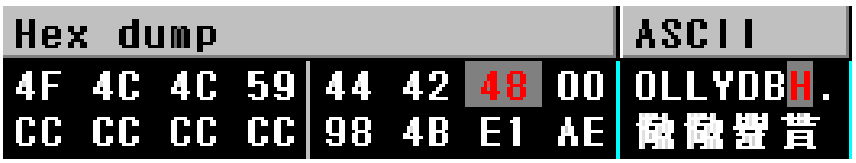
이제 디버깅 툴로 따라가보도록 하겠습니다.



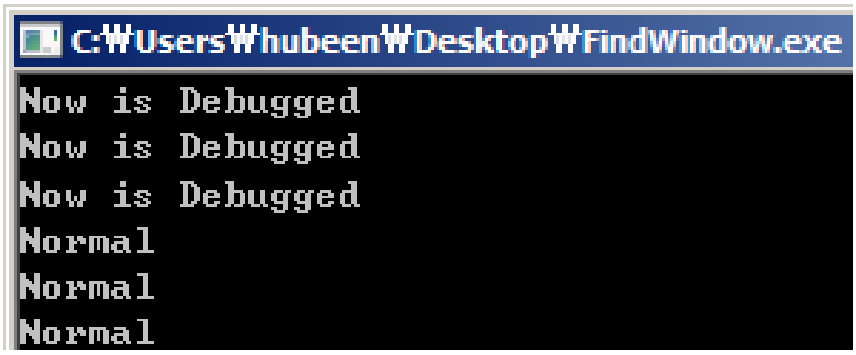
함수 안으로 들어가보면 아래와 같이 인자를 넘겨주는 것을 발견할 수 있습니다.

이를 우회하는 방법은 아래와 같습니다.





Findwindow 클래스 인자 값을 담고 있는 부분에 값을 변경을 시켜줍니다.



간단히 우회가 됐음을 볼 수 있습니다.

Reference

1. http://www.openrce.org/reference_library/anti_reversing
2. http://egloos.zum.com/anster/v/2128538
3. https://msdn.microsoft.com/ko-kr/library/windows/desktop/aa813706(v=vs.85).aspx
4. http://www.anti-reversing.com

Thanks