Anti Reverse Engineering Techniques
안티치트보안솔루션 제작

박 시율

1. 들어가며

안티 리버싱 기법을 이용하여 직접 안티치트솔루션을 간단하게 만드는걸 다뤄볼 것입니다. 세부적인 내용은 인터넷에서 쉽게 찾을 수 있으므로 간략하게 하는게 목표입니다. 또한 흔히 접해볼수있는 안티치트들인 XIGNCODE, HACKSHIELD, GAMEGUARD 에는 백분의 일도 못미치는 성능일 예정입니다.

2. 리버싱

Reverse Engineering 을 간략하게 줄여 Reversing으로 흔히 말합니다. 뜻 그대로 역으로 분석을 하는 것이며, 타겟이 되는 대상은 exe, dll, sys파일 등이 됩니다, 이런 실행파일들을 대상으로 코드를 분석해서 프로그램이 어떤역할을 하는지 함수들이 어떤기능을 하는지 알아내는 작업을 합니다. 분석을 위해서는 프로그램들은 각각 다른언어로 컴파일이 되었을테니 어셈블리어를 알아두어야 합니다.

3. 안티 리버싱

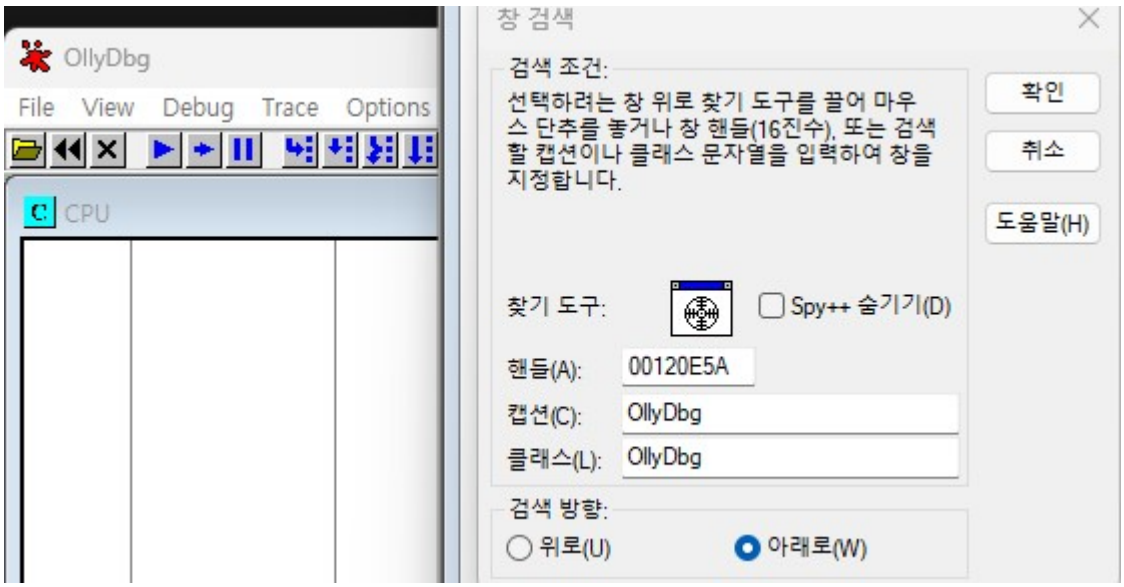
Anti - Reversing은 말 그대로 프로그램을 분석할때 분석을 어렵게 만드는 작업입니다. 코드를 난독화, 가상화하여 읽기를 어렵게 만들거나 안티 디버깅 기능을 넣어 디버깅을 어렵게 만들수 있습니다. 그래서 이러한 기법들은 온라인 게임이나 DRM이 적용된 영상, 노래 그리고 악성코드 등에 사용되며 분석을 최대한 어렵게해 방지하는 목적으로 사용됩니다. 그러나 지금 시대는 기법들이 대부분 알려져 있고 툴의 성능이 좋아졌으며, 학습할 곳도 많아져 분석자의 실력또한 높아졌습니다. 영원한 창과 방패의 싸움이지만 창이 유리한건 사실입니다. 그렇기에 뚫리지 않는 방패보단 최대한 느리게 망가지는 방패로 사용됩니다. 본 문서에서는 안티디버깅 기법을 이용하여 안티치트 솔루션을 제작할 예정입니다. 윈도우를 중심으로 할 것이며, Visual Studio를 사용해 C/C++ 언어를 사용하겠습니다.

FindWindow

FindWindow() 함수를 이용해 불법프로그램 탐지를 구현 해보겠습니다.
Visual Studio 도구에서 Spy++ 을 사용해 창들의 클래스이름과 캡션이름을 쉽게 얻을 수 있습니다.

아래는 FindWindow() 함수의 원형입니다.

```C++
HWND FindWindowA(
  [in, optional] LPCSTR lpClassName,
  [in, optional] LPCSTR lpWindowName
);
```



올리디버거의 캡션명과 클래스명을 잘 얻어 오는걸 볼 수 있습니다.클래스명과 캡션명은 항상 같지 않습니다.

우선 캡션 이름을 이용한 감지시스템을 구현해보도록 합시다.

FindWindow

```
bool WindowName(LPCSTR NameWindow)
{
    HWND WinName = FindWindowA(NULL, NameWindow);
    if ( WinName != NULL )
    {
        MessageBox(NULL, "디버거 감지 !! [FW]", "디버거 감지", MB_OK);
    }
    return true;
}

// 윈도우 이름 검색
void CheckWindow()
{
    WindowName("OllyDbg");
}

void FW_Scan()
{
    CheckWindow();
}

void DetectFW()
{
    CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(FW_Scan), NULL, 0, 0);
}
```

해당 캡션 이름이 윈도우에 존재한다면 핸들값을 반환 할 것이며, 존재하지 않는다면 NULL을 반환합니다. WinNanme이 NULL이 아니라면 핸들값이 저장되어 있단 것이고 디버거가 실행중임을 알 수 있습니다.

디버거가 실행중이라면 메세지박스를 띄워줍니다.

CreateThread를 통해 쓰레드를 만들어 FW_Scan을 실행해줍니다.

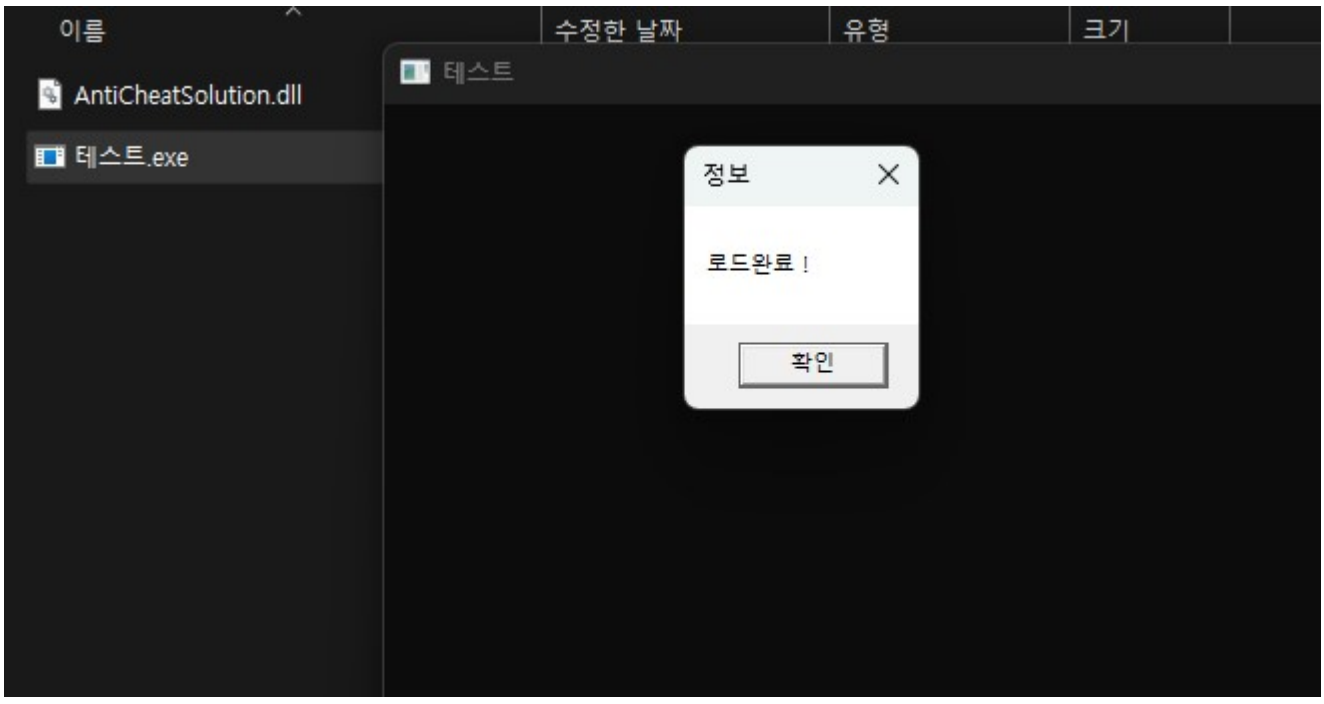
이제 테스트 해보도록 하겠습니다.

테스트를 하기 전에 우선 보호할 프로그램을 하나 만들고
안티치트를 로드하도록 하겠습니다.

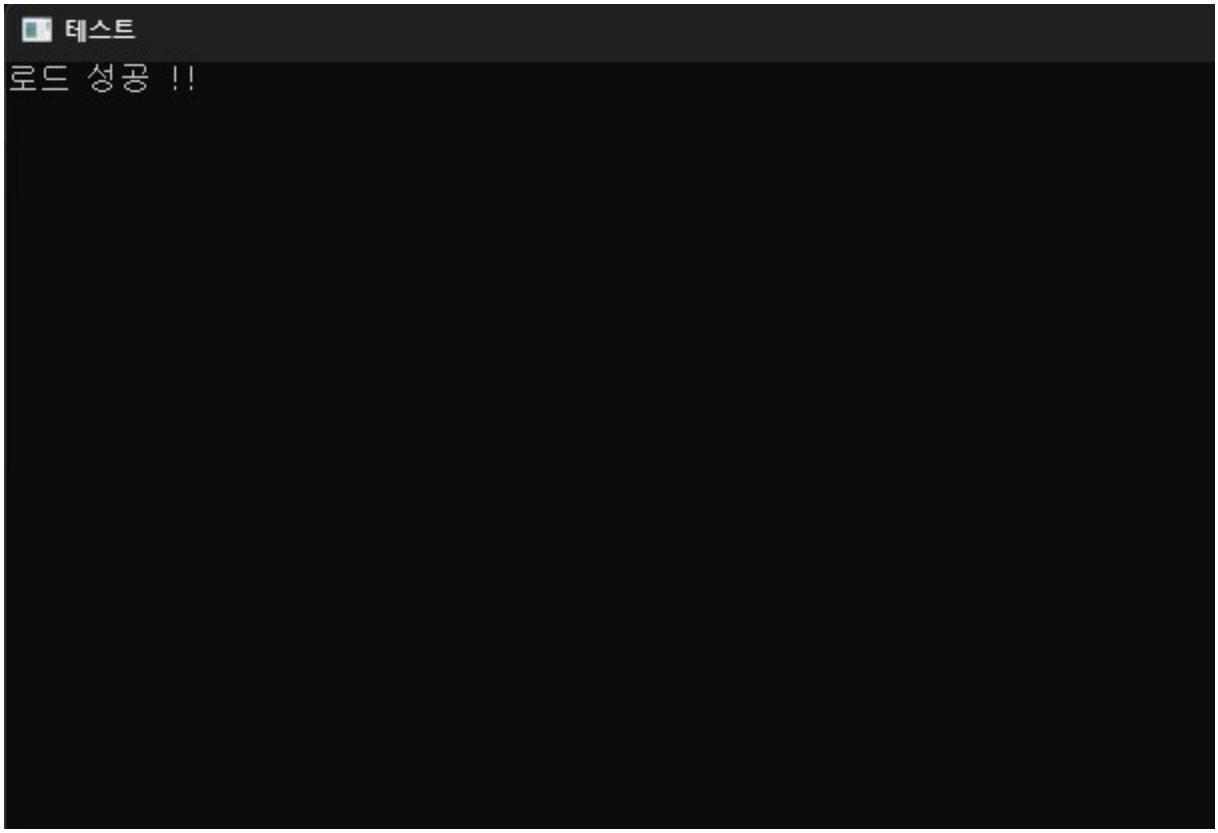
FindWindow

```cpp
#include <iostream>
#include <stdio.h>
#include <Windows.h>

using namespace std;

int main()
{
    SetConsoleTitle(TEXT("테스트"));
    HMODULE hmodule = LoadLibrary(L"AntiCheatSolution.dll");
    if (hmodule == NULL)
        cout << "로드 실패" << endl;
    else
        cout << "로드 성공 !!" << endl;

    getchar();
}
```

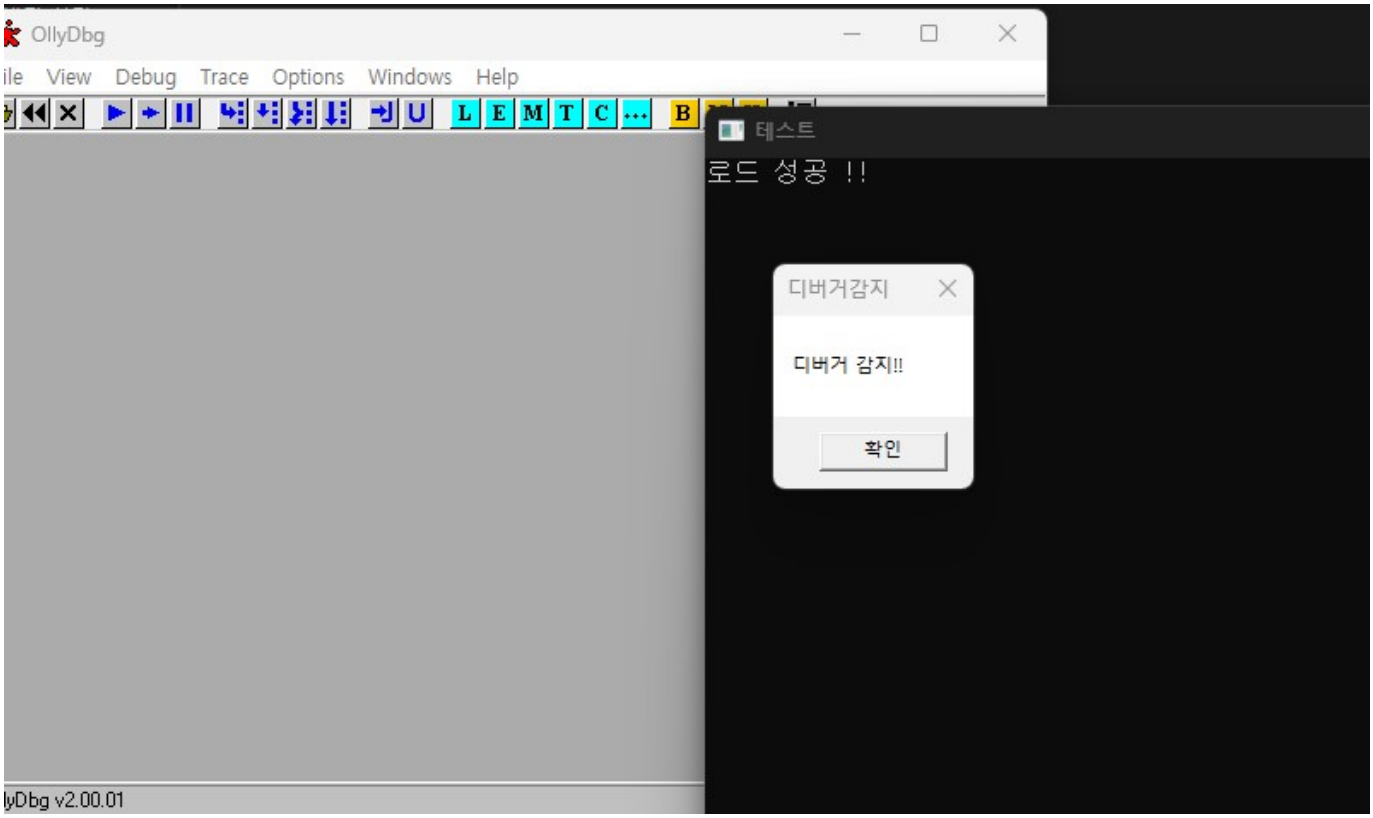
간단하게 만든 보호할 프로그램입니다.
실행하면 LoadLibrary 함수를통해 AntiCheatSolution.dll 을 불러옵니다.



FindWindow



정상적으로 로드하는 모습을 볼 수 있습니다, 이제 올리디버거를 켜서 감지를 잘 하는지 확인해 보면 디버거 감지 메세지박스를 정상적으로 띄워내는걸 볼 수 있습니다.



FindWindow

그런데 만약 프로그램의 캡션명이 무작위로 계속 바뀐다거나 사용자마다 이름이 바뀐다면 해당 방법으로는 디버거를 감지해내지 못할 것입니다. 그렇기에 이번엔 클래스 이름을 이용하여 감지해보겠습니다.

```
void ClassName(LPCSTR NameClass)
{
    HWND WinClass = FindWindowExA(NULL, NULL, NameClass, NULL);
    if (WinClass != NULL)
    {
        MessageBox(NULL, "디버거 감지!! [CN]", "디버거감지", MB_OK);
    }

}

void CheckClass()
{
    ClassName("OllyDbg");
    ClassName("ProcessHacker");
    ClassName("PROCEXPLORER");
}

void CN_Scan()
{
    CheckClass();
}

void DetectCN()
{
    CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(CN_Scan), NULL, 0, 0);
}
```

크게 다른건 없습니다만 FindWindowExA() 함수를 사용해 봤습니다.
해당함수는 클래스명 캡션명 뿐만 아니라 특정 윈도우에 포함된 자식 윈도우를 탐색할 때 사용하는 함수입니다. NULL 값을 넣어주어 FindWindow 와 똑같이 동작하지만 나중엔 기능을 추가해줄수도 있습니다.

C++ 복사

```
HWND FindWindowExA(
  [in, optional] HWND   hWndParent,
  [in, optional] HWND   hWndChildAfter,
  [in, optional] LPCSTR lpszClass,
  [in, optional] LPCSTR lpszWindow
);
```

FindWindow

기능들을 원하면 켜고 끄고 할 수 있게 구현하였습니다.
앞으로 추가되는 기능들은 전부 ON / OFF가 가능하도록 할 생각입니다.

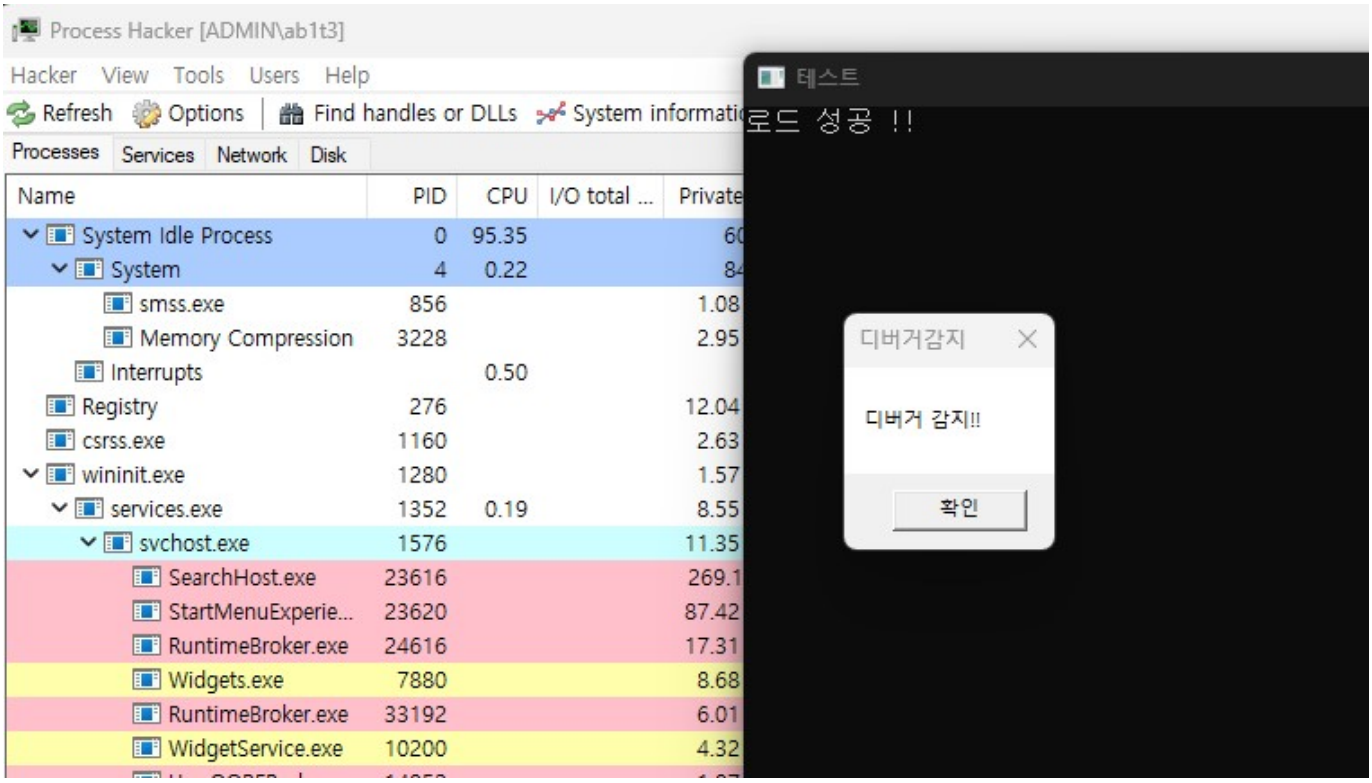
```
CONTENT anticheat;

extern "C" __declspec(dllexport) void __cdecl Siyul()
{
    // anti FindWindow
    anticheat.Detect_NameWindow = 1; // 기능 ON OFF
```

프로세스에 DLL이 어태치되면 기능 ON OFF 유무를 보고 해당 함수를 실행합니다.

그리고 해당함수는 쓰레드를 생성해 돌리도록 되어있습니다.

```
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
    Siyul();
    if (anticheat.Detect_NameWindow == 1)
    {
        DetectFW();
    }
```

FindWindow



Process Hacker 의 캡션명은 뒷쪽에 [사용자] 가 붙어있네요 이런 경우에 클래스명으로 감지를 해야하는 것이며, 잘 동작하는 모습을 볼 수 있습니다.

여담으로 프로세스해커란 프로세스를 더 자세히 들여다 볼 수 있도록 해주는 프로그램이라 보시면 됩니다. 다양한 기능들도 제공하고 있고 프로세스의 서스펜드 리슘또한 지원합니다. 꽤나 강력해서 온라인게임들은 철저히 막고 있으며 동시에 실행이 불가능 할겁니다.

ProcessName

```
void GetProcess(const char* ProcName)
{
    PROCESSENTRY32 pe32; //tlhelp32
    HANDLE hSnapshot = NULL;

    pe32.dwSize = sizeof(PROCESSENTRY32);
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (Process32First(hSnapshot, &pe32))
    {
        do
        {
            if (strcmp(pe32.szExeFile, ProcName) == 0)
            {
                MessageBoxA(NULL, "디버거 감지!! [PID]", "디버거감지", MB_OK);
            }
        } while (Process32Next(hSnapshot, &pe32));
    }
}

void CheckProcess()
{
    GetProcess("ollydbg.exe");
    GetProcess("ProcessHacker.exe");
    GetProcess("procexp64.exe");
}

void ID_Scan()
{
    CheckProcess();
}

void DetectID()
{
    CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(ID_Scan), NULL, 0, 0);
}
```

클래스명도 다르거나 변조되었다면 프로세스명으로 감지를 해야겠습니다. 이런식으로 취약점을 계속해서 막아나가는게 안티치트의 숙명이지 않을까 싶습니다.

PROCESSENTRY32 구조체를 초기화 해줍니다, CreateToolhelp32Snapshot을 이용해 스냅샷을 얻어옵니다. PROCESSENTRY32 구조체에는 szExeFile이란 녀석이 있으며 실행 파일의 이름이 들어있습니다. 미리 적어둔 디버거들의 프로세스명과 비교해 같다면 디버거 감지를 띄워줍니다.

Process Name

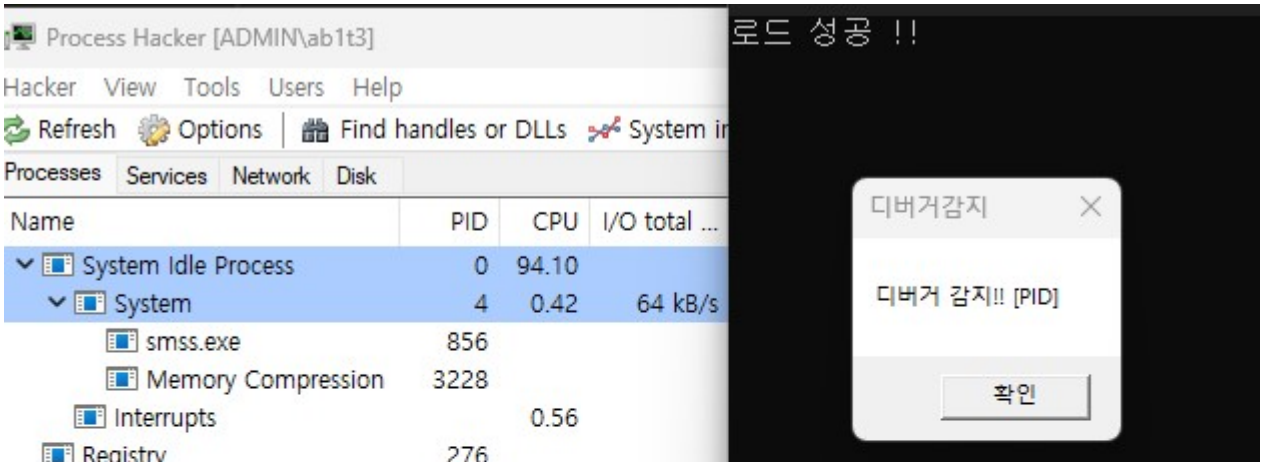
스냅샷이란 함수가 호출된 시점에 대상의 정보(프로세스 정보, 모듈, 스레드 등)를 저장해 놓은 것을 말합니다.

```C++
HANDLE CreateToolhelp32Snapshot(
  [in] DWORD dwFlags,
  [in] DWORD th32ProcessID
);
```

MSDN에서 PROCESSENTRY32 구조체에 대한 설명을 보면 친절히
스냅샷이 수행되었을 때 주소 공간에 있는 프로세스 목록을 설명한다고 적혀있습니다.
또한 dwSize를 sizeof(PROCESSENTRY32)로 설정하지 않으면 Process32First가 실패한다고 적혀있습니다. Process32First는 스냅샷이 발생한 첫 번째 프로세스에 대한 정보를 검색합니다.

```C++
typedef struct tagPROCESSENTRY32 {
  DWORD     dwSize;
  DWORD     cntUsage;
  DWORD     th32ProcessID;
  ULONG_PTR th32DefaultHeapID;
  DWORD     th32ModuleID;
  DWORD     cntThreads;
  DWORD     th32ParentProcessID;
  LONG      pcPriClassBase;
  DWORD     dwFlags;
  CHAR      szExeFile[MAX_PATH];
} PROCESSENTRY32;
```

정상적으로 감지해내는 모습을 볼 수 있습니다.



CRC Check

CRC를 통해 파일변조가 일어났는지 확인해보도록 하겠습니다. 우선 CRC란 Cyclic Redundancy Check의 약자로, 무결성을 체크하는 방법 중 하나입니다. 원본 데이터가 멀쩡하다는걸 증명하기 위해 사용합니다. 그 과정에서, 여러 종류의 다항식이 쓰일 수 있으며 널리 사용되는 CRC다항식인 CRC-32 를 이용하겠습니다.

CRC32의 다항식(Polynomial)은 아래와 같습니다.

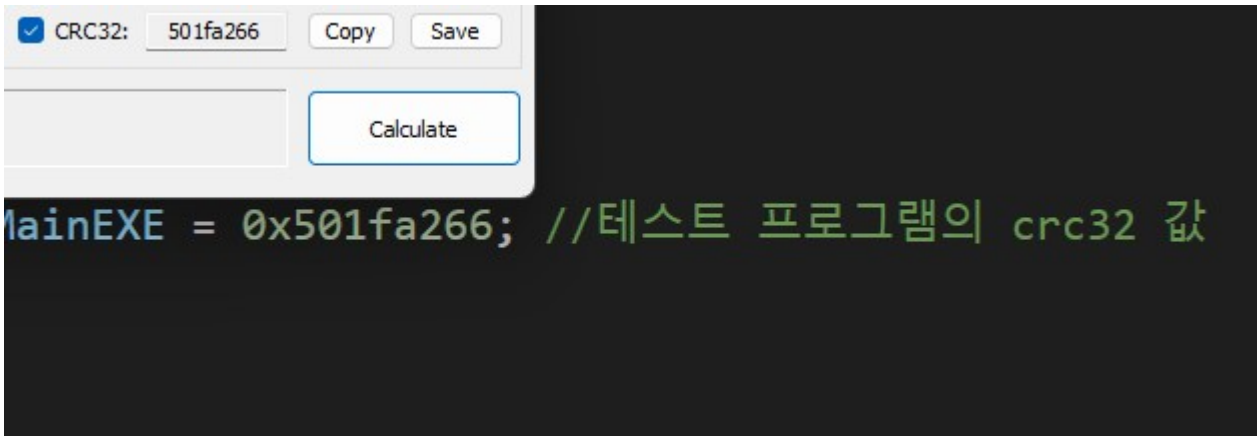
```
// 다항식 : x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
// 32/31/30/29/28/27/26/25/24/23/22/21/20/19/18/17/16/15/14/13/12/11/10/9/8/7/6/5/4/3/2/1/0
//  1/ 0/ 0/ 0/ 0/ 0/ 1/ 0/ 0/ 1/ 1/ 0/ 0/ 0/ 0/ 0/ 1/ 0/ 0/ 0/ 1/ 1/ 1/0/1/1/0/1/1/0/1/1/1
#define CRC32_POLYNOMIAL 0x04c11db7 //crc32 다항식을 hex & reverse = edb88320
#define CRC32BUFSZ 1024
```

다항식을 16진수로 표현하려면 주석에 적어둔 것처럼 x의 차수에 해당하는 비트에 1을 쓰고 나머진 0을 씁니다. 그럼 2진수로 나오고 위의 주석처럼 되는데 31번째 비트부터 읽게되면
0x4C11DB7 이 나오며 0번째비트부터 거꾸로 읽게되면 0xEDB88320 이 나오는 것입니다.

```
CheckSum Check;
void CheckSum::DetectCRC()
{
    Check.CRCInit();
    if (anticheat.CRC_Main == 1)
    {
        long Crc1 = Check.FileCRC(anticheat.ProtectEXE);
        if (Crc1 != anticheat.CRC_MainEXE)
        {
            MessageBoxA(NULL, "파일변조감지 !!", "위험", MB_OK);
        }
    }
}

void MainCRC(void* lpParam)
{
    Check.DetectCRC();
}
```

미리 구해둔 보호할 프로그램의 CRC32 값을 안티치트솔루션에 입력해둔후 CRC검사를 했을때 만약 보호프로그램이 변조가 일어났다면 자연스레 CRC32 값이 바뀌게 됩니다.
그렇다면 해쉬값이 맞지 않게되면서 파일변조감지가 일어나게 됩니다.

CRC Check



보호할 프로그램의 CRC32값을 구해 넣어주었습니다. 값이 달라지게 된다면, 변조를 감지합니다.

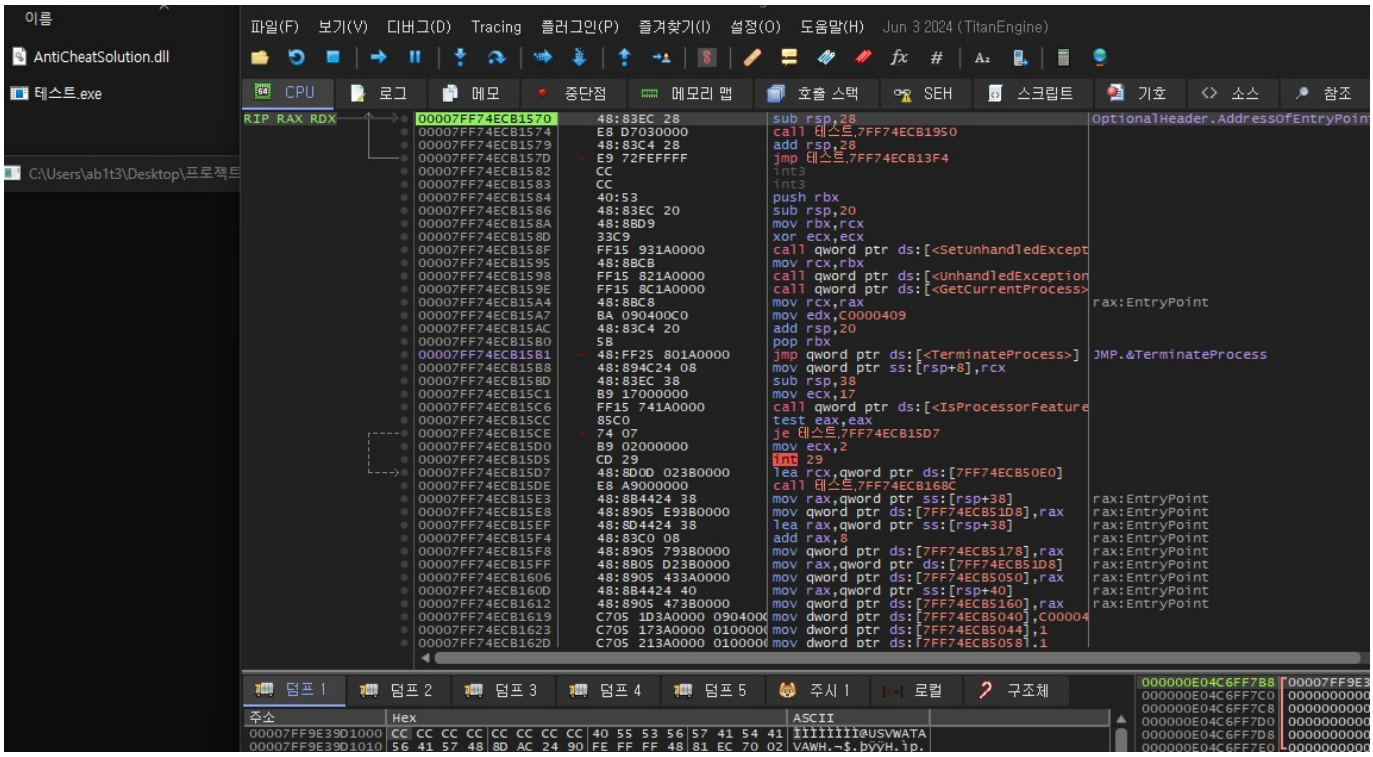
```
// EXE Data
anticheat.ProtectEXE = "테스트.exe";

// anti FindWindow
anticheat.Detect_NameWindow = 1; // 기능 ON OFF
// anti ClassName
anticheat.Detect_NameClass = 1;
// anti ProcessID
anticheat.Detect_ProcID = 1;
// CRC Check
anticheat.CRC_Main = 1;
anticheat.CRC_MainEXE = 0x501fa266; //테스트 프로그램의 crc32 값
```

프로세스에 딜이 어태치되면 아래가 실행됩니다. beginthread는 표준함수가 안전하게 실행됩니다.

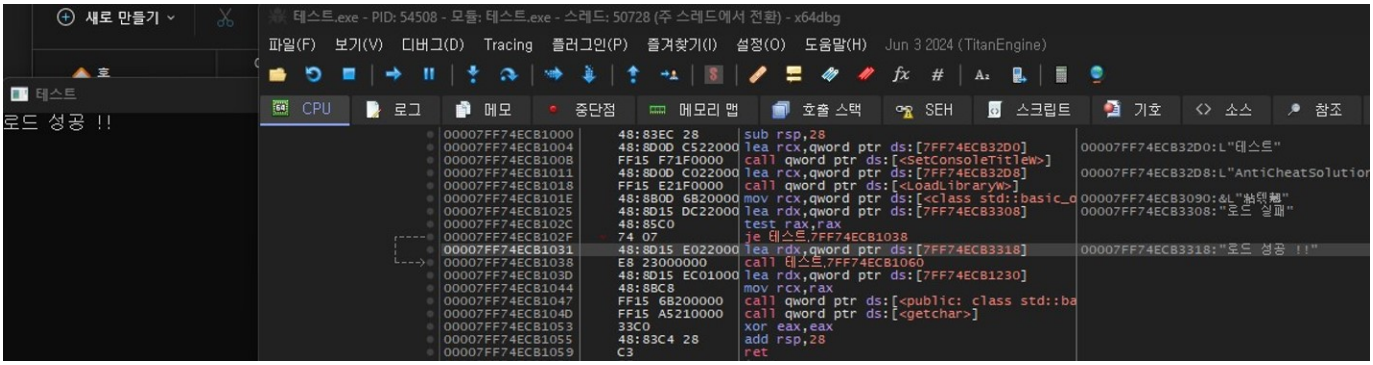
```
    if (anticheat.Detect_ProcID == 1)
    {
        DetectID();
    }
    if (anticheat.CRC_Main == 1)
    {
        _beginthread(MainCRC, 0, NULL);
    }
    MessageBoxA(0, "로드완료 !", "정보", MB_OK);
```

CRC Check

테스트를 위해 보호할 프로그램을 변조해보겠습니다. 디버거를 통해 테스트 프로그램의 LoadLibrary로 AntiCheatSolution.dll 을 성공적으로 로드했을때 출력되는 "로드 성공 !!" 문장을 제 임의로 바꾸도록 하겠습니다.



디버거로 테스트 프로그램을 열어주었습니다. 그리고 main 함수 부분을 찾아 가보겠습니다.

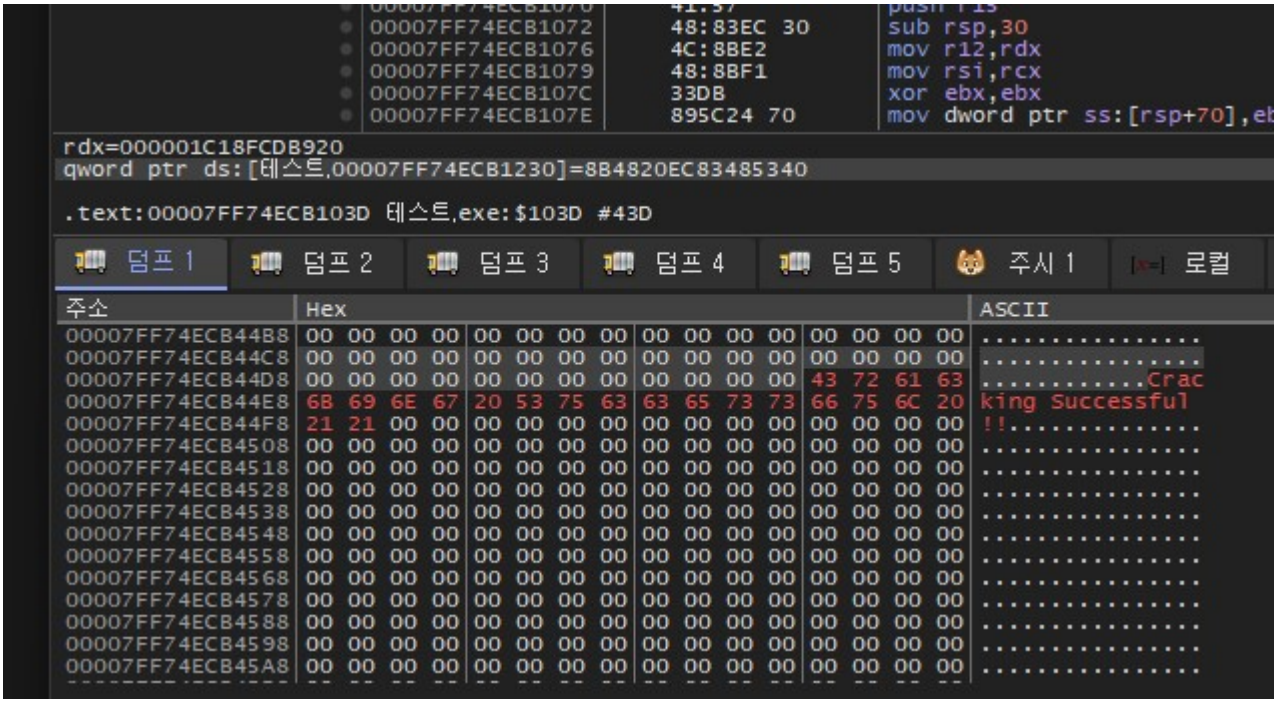


테스트 프로그램의 캡션명도 call qword ptr ds:[<SetConsoleTitleW>] 가 실행되며 바뀐 모습을 볼 수 있습니다. 클릭해둔 로드 성공 !! 의 바로 밑인 call 부분은 cout 함수를 call 하는 부분이겠군요

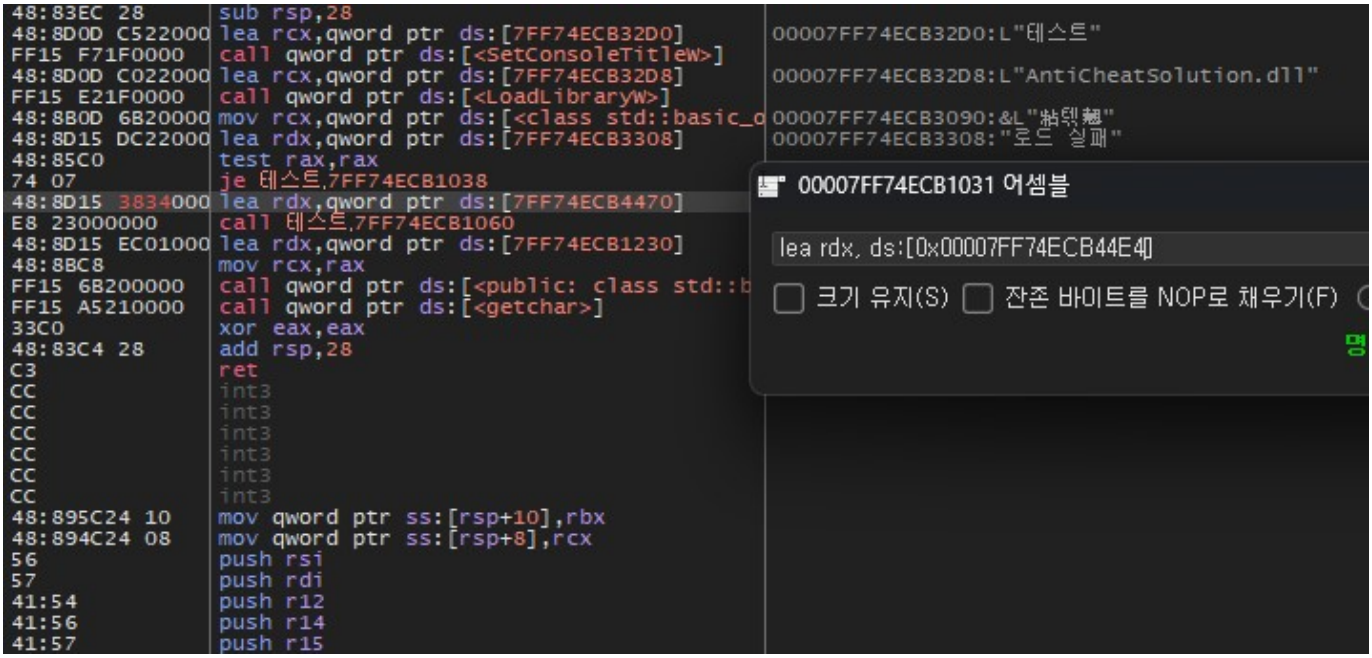
lea rdx, qword ptr ds: [7FF74ECB3318] 을 보게되면 7FF74ECB3318 주소에 가면 로드 성공 !! 이라는 글자가 저장되어 있겠네요, 그런데 바꾸려면 로드 성공 !! 의 크기에 맞게 바꿀수만 있습니다.
더 긴 글자로 바꾸려면 코드케이브 공간을 이용하면 됩니다.

CRC Check

코드 케이브 공간은 아래와 같이 00 00 00 으로만 이루어진 공간입니다.
PE View 로 파일을 열어 코드케이브 공간을 확인할 수 있습니다만 그냥 대충 내려서 빈공간에 끄적여 보겠습니다.



빈 공간에 Cracking Successful !! 이란 단어를 저장해 주었습니다. 이제 이 문구를 출력하게하면 되겠네요



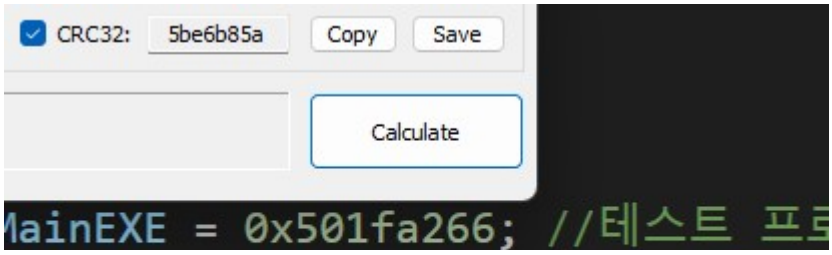
main 함수 위치로 돌아가서 "로드 성공!!" 이 들어 있던 주소를, 제가 따로 저장해둔 문구의 주소로 변경해줍니다. 7FF74ECB4470 -> 7FF74ECB44E4

CRC Check

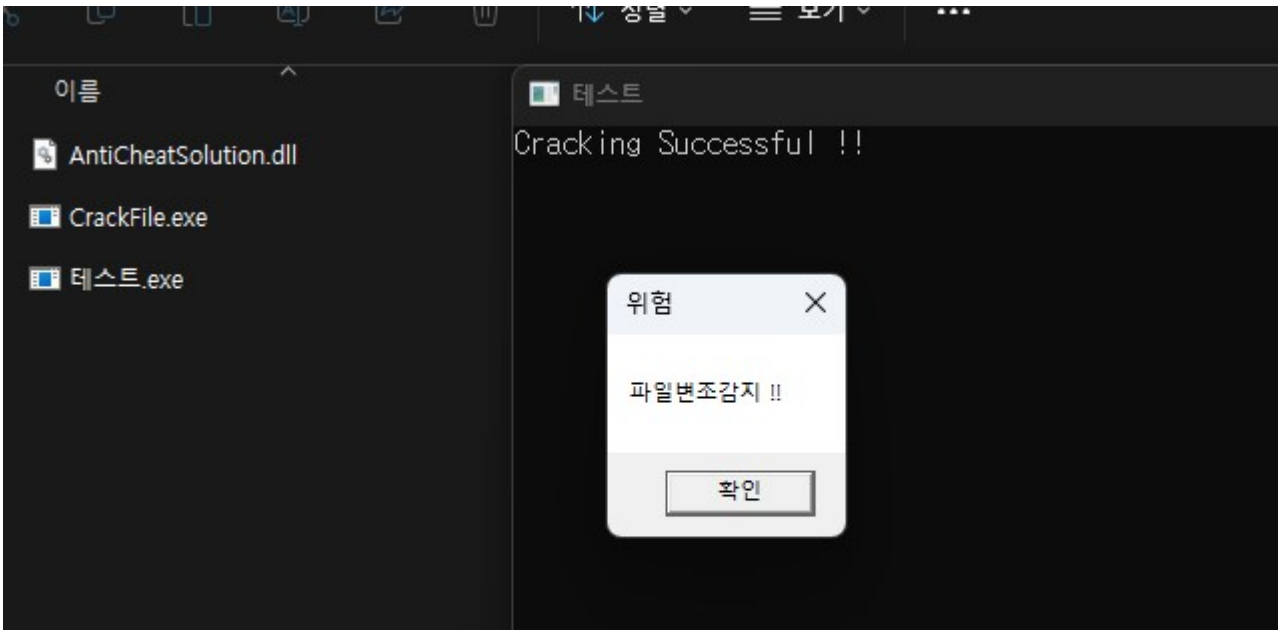
```
83EC 28          sub rsp,28
8D0D C5220000    lea rcx,qword ptr ds:[7FF74ECB32D0]      00007FF74ECB32D0:L"테스트"
15 F71F0000      call qword ptr ds:[<SetConsoleTitleW>]
8D0D C0220000    lea rcx,qword ptr ds:[7FF74ECB32D8]      00007FF74ECB32D8:L"AntiCheatSolution.dll"
15 E21F0000      call qword ptr ds:[<LoadLibraryW>]
8B0D 6B200000    mov rcx,qword ptr ds:[<class std::basic_o 00007FF74ECB3090:&L"鮎뎅鶫"
8D15 DC220000    lea rdx,qword ptr ds:[7FF74ECB3308]      00007FF74ECB3308:"로드 실패"
85C0             test rax,rax
07               je 테스트.7FF74ECB1038
8D15 AC340000    lea rdx,qword ptr ds:[7FF74ECB44E4]      00007FF74ECB44E4:"Cracking Successful !!"
23000000         call 테스트.7FF74ECB1060
8D15 EC010000    lea rdx,qword ptr ds:[7FF74ECB1230]
8BC8             mov rcx,rax
15 6B200000      call qword ptr ds:[<public: class std::ba
15 A5210000      call qword ptr ds:[<getchar>]
30               xor eax,eax
83C4 28          add rsp,28
                 ret
                 int3
                 int3
                 int3
```

Cracking Successful !! 로 잘 변경 되었습니다. 이렇게 변조한 프로그램을 따로 저장해두고 실행해서 과연 파일변조를 잘 감지하는지 확인하면 됩니다.

보통 보호할 프로그램들은 패킹이 되어있습니다 그렇기에 메인함수를 찾아가는건 물론
위처럼 text또한 제대로 나오지 않습니다. 실행중인 프로그램조차 치트엔진으로 값을 변경하게 되면 crc가 돌아가고 있어 변조를 감지해냅니다.



제 문구를 임의로 패치한 프로그램은 crc32값이 바뀌었고 실행해보면 로드 성공이라 뜨던 문구는 제가 패치한 Cracking Successful !! 로 바뀌어 출력이 되는 모습을 볼 수 있으며, 파일변조감지를 해내는 모습을 볼 수 있습니다.



INT 3

INT 3 명령어는 Software BreakPoint 로 많이 사용되는 BreakPoint중 하나입니다.
만약 디버깅 중이라면 이 명령어가 실행될 때 브레이크포인트가 트리거 되면서 자연스럽게 브레이크포인트로 인식해 넘어가지만 아닐 때는 예외가 발생하며 예외 구문에 걸리게 됩니다. 올리디버거에서 브레이크포인트를 걸 경우 Opcode는 CC로 설정됩니다. INT 3가 실행될 경우 CC로 설정이 됩니다.

```
void SoftwareBreakPoint()
{
    int flag = 0;
    __try
    {
        __asm
        {
            int 3;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        flag = 1;
    }
    if (flag == 0)
        MessageBox(NULL, "디버깅 감지 !! [int3]", "SBP", MB_OK);
}
```

INT 41 명령어는 커널 디버거를 탐지할 때 사용됩니다. 보통 이 명령어는 유저모드(ring 3)에서 실행이 되지 않습니다. 만약 실행을 하게 되면 int d가 수행되어 예외가 발생하게 됩니다.

```
void SoftwareBreakPoint2()
{
    int flag = 0;
    __try
    {
        __asm
        {
            int 0x41
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        flag = 1;
    }
    if (flag ==0)
        MessageBox(NULL, "디버깅 감지 !! [int41]", "SBP", MB_OK);
```

HardwareBreakPoint

Hardware BreakPoint 는 브레이크 포인트 횟수 제한이 있지만 코드변경이 없습니다.
디버그 레지스터8개(Dr0 ~ Dr7) 를 이용하여 cpu레벨에서 설정합니다.
dr0 ~ dr3는 브레이크포인트 주소를 담는데 사용되고, Dr4, Dr5 는 다른목적을 위해 예약된 레지스터, Dr6, Dr7은 브레이크포인트 행동을 제어하기 위해 사용됩니다. 그렇기에, Dr0 ~ Dr3 레지스터를 확인해 값이 들어있으면 하드웨어브레이크포인트가 설정된 것입니다.
코드변조가 없기때문에 crc체크를 하는 프로그램에서 하드웨어브레이크 포인트를 걸어 eip 레지스터를 바꾸어 핵을 사용할때 사용됩니다.

```
void HardwareBreakPoint()
{
    int flag = 0;
    CONTEXT my;

    ZeroMemory(&my, sizeof(CONTEXT));
    my.ContextFlags = CONTEXT_DEBUG_REGISTERS;

    if (GetThreadContext(GetCurrentThread(), &my) != 0)
    {
        if (my.Dr0 != 0)
            ++flag;
        if (my.Dr1 != 0)
            ++flag;
        if (my.Dr2 != 0)
            ++flag;
        if (my.Dr3 != 0)
            ++flag;
    }

    if (flag != 0)
        MessageBox(NULL, "디버깅감지 !! [HBP]", "HBP", MB_OK);
}
```

BeingDebugged

BeingDebugged 함수는 TEB(Thread Environment Block)와 PEB(Process Environment Block)를 사용하며, PEB구조체에 BeingDebugged flag를 검사해 디버깅을 확인 후 디버깅 중이면 true 아니라면 false를 반환한다.

C++ 복사

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

```
void BeingDebugged()
{
    if (IsDebuggerPresent())
        MessageBox(NULL, "디버깅감지 !! [BD]", "BD", MB_OK);
}
```

NtQuerySystemInformation

NtQuerySystemInformation은 현재 운영체제의 정보들을 가져오는 함수로 운영체제가 디버그 모드로 부팅 되었는지 아닌지를 판별하는 기법으로 주로 사용된다. 해당 클래스의 35번째에 존재하는 SystemKernelDebuggerInformation(0x23)을 이용하면 SYSTEM_KERNEL_DEBUGGER_INFORMATION 구조체의 DebuggerEnabled 값을 확인할 수 있다. EAX레지스터에 값이 반환되며 al에 KdDebuggerEnabled 값이 들어가고 ah에 KdDebuggerNotPresent 값이 들어간다, ah에 0이 들어가 있으면 디버깅 중이란 뜻이다.

```
typedef struct _SYSTEM_KERNEL_DEBUGGER_INFORMATION
{
    BOOLEAN KdDebuggerEnabled;
    BOOLEAN KdDebuggerNotPresent;
} SYSTEM_KERNEL_DEBUGGER_INFORMATION;

typedef NTSTATUS(WINAPI* ntqsi)(ULONG, PVOID, ULONG, PULONG);

void NtQuerySystem()
{
    SYSTEM_KERNEL_DEBUGGER_INFORMATION sk;
    ntqsi qsi = (ntqsi)GetProcAddress(GetModuleHandle(L"ntdll.dll"), "NtQuerySystemInformation");
    qsi(0x23, &sk, 2, NULL);
    if (sk.KdDebuggerEnabled && !sk.KdDebuggerNotPresent)
        MessageBox(NULL, "디버깅감지 !!", "NtQSI", MB_OK);
}
```

난독화, Anti Dump

- Dummy Codes
  더미코드는 아무런 역할을 하지 않는 코드모음으로, 사이사이에 삽입해 난독화를 시키거나 분석 툴이 인스트럭션들을 잘못 해석하게 하는 용도로 사용된다

```
#define JUNK_CODE_ONE   \
    __asm{push eax}     \
    __asm{xor eax, eax} \
    __asm{setpo al}     \
    __asm{push edx}     \
    __asm{xor edx, eax} \
    __asm{sal edx, 2}   \
    __asm{xchg eax, edx}\
    __asm{pop edx}      \
    __asm{or eax, ecx}  \
    __asm{pop eax}
```

- Anti Dumping
  - Erase PE Header
  메모리 상에 존재하는 PE Header를 지워 덤핑을 하는 과정을 방해할 수 있다.

```
void ErasePE()
{
    DWORD pe = 0;
    PCHAR base = (PCHAR)GetModuleHandle(NULL);

    VirtualProtect(base, 4096, PAGE_READWRITE, &pe);
    memset(base, 0, 4096);
}
```

Packing

- Packing
  프로그램을 보호하는 방법으로 패킹이란 방법이 있습니다. 패킹이란 실행파일을 암호화 및 압축을 하는 방법을 말합니다. 압축하고 암호화를 할 때 다른 섹션을 만들어 그 곳에 복호화 하는 알고리즘을 저장하며 실행 시 암호화된 데이터를 복호화 한 후 실행시킵니다.

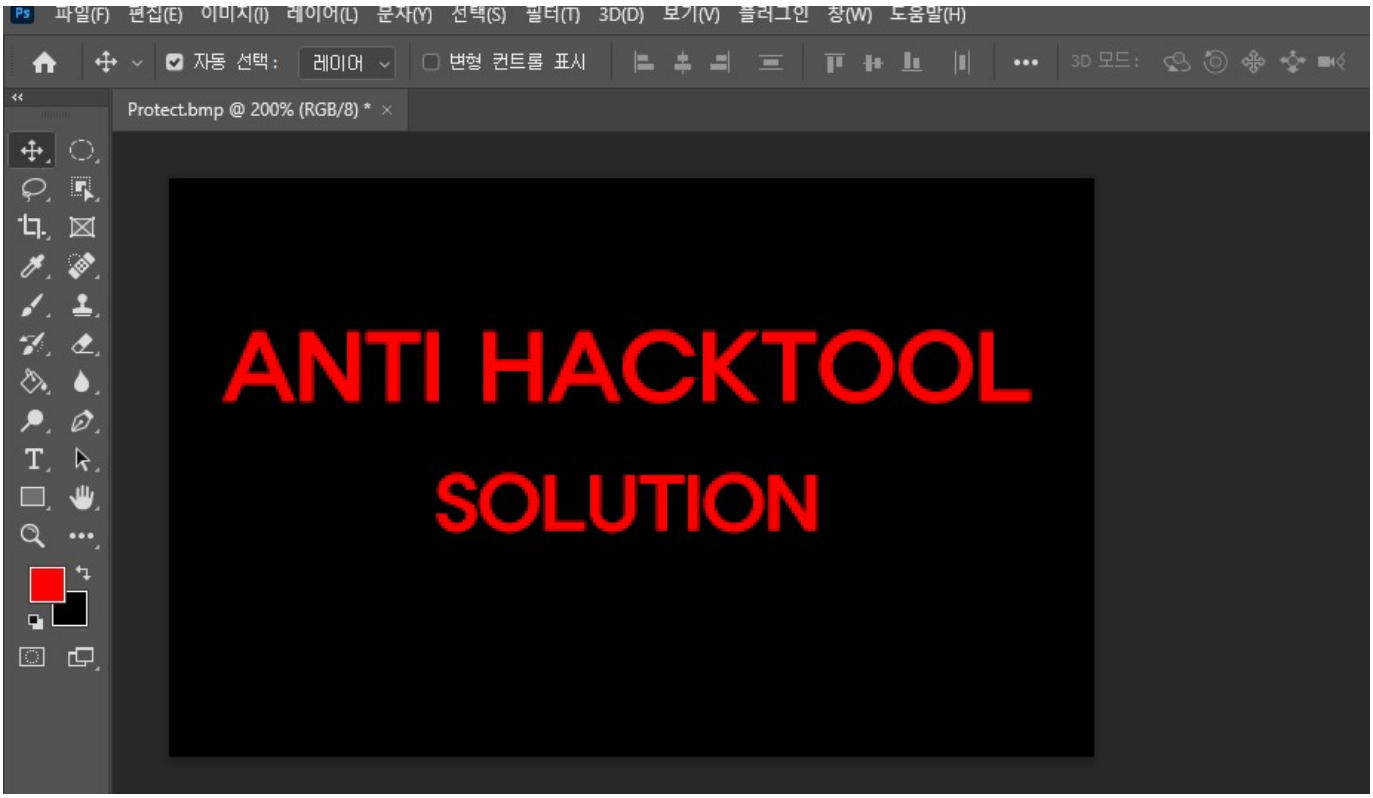
- Themida & Winlicense
  현존하는 제일 강력한 패커들이며 anti virtual machine, anti dumping, anti debugging, EntryPoint 섞기, Resources 암호화 등 수많은 기능들을 제공하며 더미다 제작사에서는 라이센스 관리기능을 추가한 윈라이센스또한 출시했다. 많은 상용프로그램들에 사용되고 있으며 카카오톡 같은 프로그램들도 Themida 패커를 사용하고 있다.

- UPX
  가벼우면서 포맷을 다양하게 지원하고 프리웨어이다. 패킹 시 upx0, upx1 섹션이 생성되고 upx0에는 섹션및 기타 정보초기화 관련 코드가 존재하며 upx1에는 복호화 알고리즘이 존재한다. 많이 쓰이는 패커 중 하나이다, UPX Unpacker 도 존재해 손쉽게 언패킹이 가능하다.

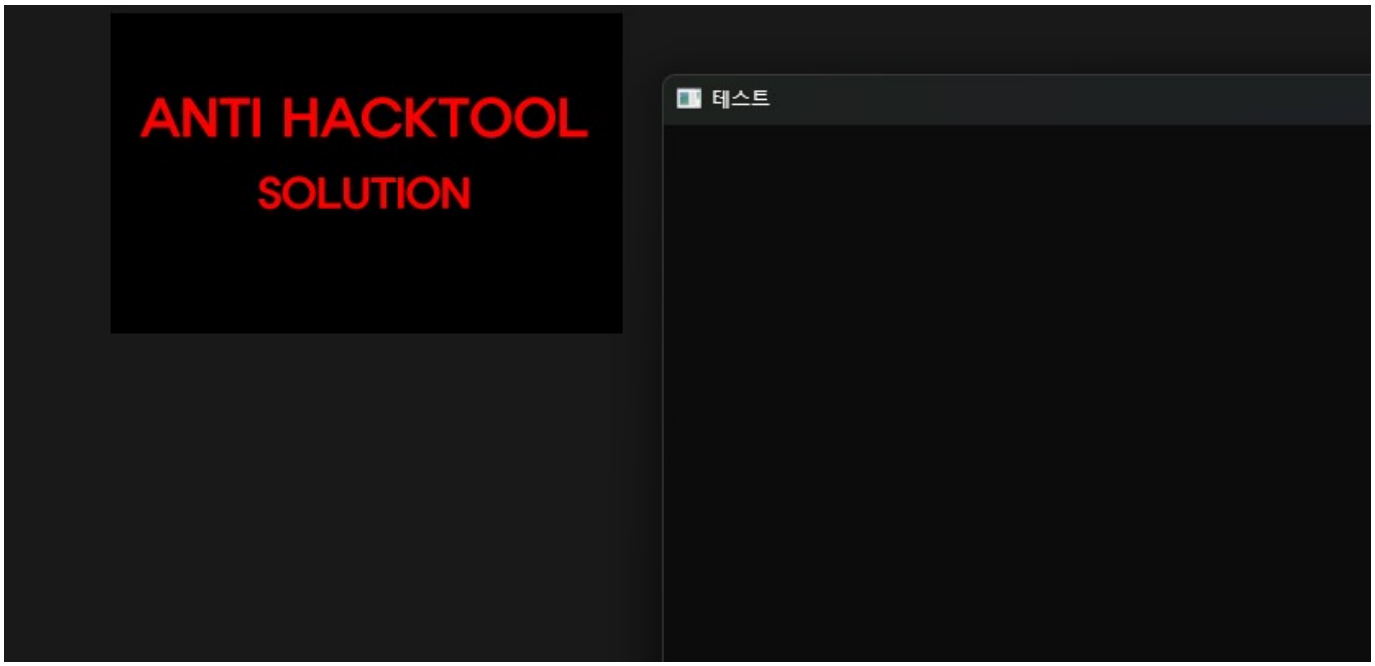
- ASProtect
  코드 압축률은 좋지 못하나 암호화는 잘 되어있다. Stolen Bytes 기법이 최초로 구현된 패커이다. 상용 프로그램이다.

- Stolen Bytes
OEP 주변의 몇 개의 인스트럭션들을 분리된 메모리에 따로 복사한 후 실행하는 기법을 말한다. 아무곳에서나 인스트럭션을 복사해 실행할 수는 없고 사이즈를 미리 알고 있어야 예외없이 실행가능하다.

상용 보안솔루션들은 실행이 될 때 스플래시 이미지가 뜨며 로딩을 알린다.
본 문서에서 또한 스플래시 이미지를 넣어주어 조금 있어보이는 모습을 만들기 위해 포토샵으로 대충 만들어 보았습니다..



이제 AntiCheatSolution.dll 이 로드 될때 스플래시 이미지가 뜨는 모습을 볼 수 있습니다.



▪ 끝맺으며

지금까지 몇 가지의 안티덤핑, 패킹, 안티디버깅 기법들을 알아보았습니다.
물론 훨씬 많은 기법들이 존재하며 커널권한에서 동작하는 드라이버를 통한 보호기법도
존재합니다. 예를들어 커널함수인 ObRegisterCallbacks를 통해 메모리를 보여주지 않는다던가 하는 기술말이죠 이러한 드라이버들은 커널 ring0 권한에서 놀기때문에 상당히 치명적입니다.
그렇기 때문에 서명되지 않은 드라이버는 설치하는 과정또한 번거롭습니다.
또한 CreateService 함수를 통해 드라이버를 서비스에 등록하고 StartService로 서비스를 시작해주어야 동작합니다.

너무나도 미흡한 보안솔루션을 만들어 보았습니다만, 지식이 늘고 기술이 좋아지면 다음번엔
더욱 많은 안티디버깅 기법들, 드라이버도 건드려서 더욱 촘촘한 보안솔루션을 만들어보고 싶은마음이 드네요

많이 부족하고 글 쓰는 재주가 없지만 잘 봐주셨으면 감사하겠습니다.

dllmain.cpp

```
#include "MainH.h"
CONTENT anticheat;

extern "C" __declspec(dllexport) void __cdecl Siyul()
{
	// EXE Data
	anticheat.ProtectEXE = "테스트.exe";
	// Splash Image
	anticheat.Splash_Activate = 1;

	// anti FindWindow
	anticheat.Detect_NameWindow = 1; // 기능 ON OFF
	// anti ClassName
	anticheat.Detect_NameClass = 1;
	// anti ProcessID
	anticheat.Detect_ProcID = 1;
	// CRC Check
	anticheat.CRC_Main = 1;
	anticheat.CRC_MainEXE = 0x501fa266; //테스트 프로그램의 crc32 값
	// anti SoftWare BreakPoint - int 3
	anticheat.Detect_SBP = 0;
	// anti Hardware BreakPoint
	anticheat.Detect_HBP = 1;
	// anti BD
	anticheat.Detect_BD = 1;
	// anti NtQuerySystemInformation
	anticheat.Detect_NtQSI = 1;
	// anti Dump - Erase PE Header
	anticheat.Anti_ErasePE = 1;

}

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
   switch (ul_reason_for_call)
   {
   case DLL_PROCESS_ATTACH:
                        Siyul();
                        if (anticheat.Splash_Activate == 1)
                        {
                                SplashShow();
                        }
                        if (anticheat.Detect_NameWindow == 1)
                        {
                                DetectFW();
                        }
                        if (anticheat.Detect_NameClass == 1)
                        {
                                DetectCN();
                        }
                        if (anticheat.Detect_ProcID == 1)
                        {
                                DetectID();
                        }
```

dllmain.cpp

```
if (anticheat.CRC_Main == 1)
		{
			_beginthread(MainCRC, 0, NULL);
		}
		if (anticheat.Detect_SBP == 1)
		{
			DetectSBP();
		}
		if (anticheat.Detect_HBP == 1)
		{
			DetectHBP();
		}
		if (anticheat.Detect_BD == 1)
		{
			DetectBD();
		}
		if (anticheat.Detect_NtQSI == 1)
		{
			DetectNtQSI();
		}
		if (anticheat.Anti_ErasePE == 1)
		{
			AntiPE();
		}
		MessageBoxA(0, "로드완료 !", "정보", MB_OK);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

MainH.h

```
#include <windows.h>
#include <string>
#include <stdio.h>
#include <TlHelp32.h>
#include <process.h>
#include "Aclass.h"
#include "Bclass.h"
#include "CRC.h"

#pragma warning (disable: 4996)

using namespace std;

extern  "C"  __declspec(dllexport) void __cdecl Siyul();
```

Aclass.h

```
class CONTENT
{
public:
	const char* ProtectEXE;

	int Splash_Activate;
	int Detect_NameWindow;
	int Detect_NameClass;
	int Detect_ProcID;
	int CRC_Main;
	int Detect_SBP;
	int Detect_HBP;
	int Detect_BD;
	int Detect_NtQSI;
	int Anti_ErasePE;

	long CRC_MainEXE;

};
extern CONTENT anticheat;
```

Bclass.h

```
void DetectFW();
void DetectCN();
void DetectID();
void MainCRC(void* lpParam);
void DetectSBP();
void DetectHBP();
void DetectBD();
void DetectNtQSI();
void SplashShow();
void AntiPE();
```

CRC.h

```
#ifndef CHECKSUM_H
#define CHECKSUM_H


// 다항식 : x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
// 32/31/30/29/28/27/26/25/24/23/22/21/20/19/18/17/16/15/14/13/12/11/10/9/8/7/6/5/4/3/2/1/0
//  1/ 0/ 0/ 0/ 0/ 0/ 1/ 0/ 0/ 1/ 1/ 0/ 0/ 0/ 0/ 0/ 1/ 0/ 0/ 0/ 1/ 1/ 1/0/1/1/0/1/1/0/1/1/1
#define CRC32_POLYNOMIAL 0x04c11db7 //crc32 다항식을 hex & reverse = edb88320
#define CRC32BUFSZ 1024

class CheckSum {

public:

	void CRCInit(void);
	void DetectCRC();
	unsigned long FileCRC(const char* sFileName);
	unsigned long FullCRC(unsigned char* sData, unsigned long ulLength);
	void PartialCRC(unsigned long* ulInCRC, unsigned char* sData, unsigned long ulLength);

private:

	unsigned long Reflect(unsigned long ulReflect, char cChar);
	unsigned long ulTable[256];
};

extern CheckSum Check;

#endif
```

Detect_Window.cpp

```
#include "MainH.h"

bool WindowName(LPCSTR NameWindow)
{
	HWND WinName = FindWindowA(NULL, NameWindow);
	if ( WinName != NULL )
	{
		MessageBox(NULL, "디버거 감지 !! [FW]", "디버거 감지", MB_OK);
	}
	return true;
}

// 윈도우 이름 검색
void CheckWindow()
{
	WindowName("OllyDbg");
}

void FW_Scan()
{
	CheckWindow();
}

void DetectFW()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(FW_Scan), NULL, 0, 0);
}
```

Detect_Class.cpp

```
#include "MainH.h"

void ClassName(LPCSTR NameClass)
{
	HWND WinClass = FindWindowExA(NULL, NULL, NameClass, NULL);
	if (WinClass != NULL)
	{
		MessageBox(NULL, "디버거 감지!! [CN]", "디버거감지", MB_OK);
	}

}

void CheckClass()
{
	ClassName("OllyDbg");
	ClassName("ProcessHacker");
	ClassName("PROCEXPLORER");
}

void CN_Scan()
{
	CheckClass();
}

void DetectCN()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(CN_Scan), NULL, 0, 0);
}
```

Detect_PID.cpp

```
#include "MainH.h"

void GetProcess(const char* ProcName)
{
	PROCESSENTRY32 pe32; //tlhelp32
	HANDLE hSnapshot = NULL;

	pe32.dwSize = sizeof(PROCESSENTRY32);
	hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

	if (Process32First(hSnapshot, &pe32))
	{
		do
		{
			if (strcmp(pe32.szExeFile, ProcName) == 0)
			{
				MessageBoxA(NULL, "디버거 감지!! [PID]", "디버거감지", MB_OK);
			}
		} while (Process32Next(hSnapshot, &pe32));
	}
}

void CheckProcess()
{
	GetProcess("ollydbg.exe");
	GetProcess("ProcessHacker.exe");
	GetProcess("procexp64.exe");
}

void ID_Scan()
{
	CheckProcess();
}

void DetectID()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(ID_Scan), NULL, 0, 0);
}
```

CRC.cpp

```
#include "MainH.h"

void CheckSum::CRCInit(void)
{
	memset(&this->ulTable, 0, sizeof(this->ulTable));

	for (int iCodes = 0; iCodes <= 0xFF; iCodes++)
	{
		this->ulTable[iCodes] = this->Reflect(iCodes, 8) << 24;

		for (int iPos = 0; iPos < 8; iPos++)
		{
			this->ulTable[iCodes] = (this->ulTable[iCodes] << 1) ^
				(this->ulTable[iCodes] & (1 << 31) ? CRC32_POLYNOMIAL : 0);
		}
		this->ulTable[iCodes] = this->Reflect(this->ulTable[iCodes], 32);
	}
}
unsigned long CheckSum::Reflect(unsigned long ulReflect, char cChar)
{
	unsigned long ulValue = 0;
	for (int iPos = 1; iPos < (cChar + 1); iPos++)
	{
		if (ulReflect & 1) ulValue |= 1 << (cChar - iPos);
		ulReflect >>= 1;
	}
	return ulValue;
}
unsigned long CheckSum::FileCRC(const char* sFileName)
{
	unsigned long ulCRC = 0xffffffff;
	FILE* fSource = NULL;
	unsigned char sBuf[CRC32BUFSZ];
	int iBytesRead = 0;
	if ((fSource = fopen(sFileName, "rb")) == NULL) //바이너리형식으로 파일을 읽음.
	{
		return 0xffffffff;
	}
	do {
		iBytesRead = fread(sBuf, sizeof(char), CRC32BUFSZ, fSource);
		this->PartialCRC(&ulCRC, sBuf, iBytesRead);
	} while (iBytesRead == CRC32BUFSZ);
	fclose(fSource);
	return(ulCRC ^ 0xffffffff);
}
unsigned long CheckSum::FullCRC(unsigned char* sData, unsigned long ulLength)
{
	unsigned long ulCRC = 0xffffffff;
	this->PartialCRC(&ulCRC, sData, ulLength);
	return ulCRC ^ 0xffffffff;
}
void CheckSum::PartialCRC(unsigned long* ulInCRC, unsigned char* sData, unsigned long ulLength)
{
	while (ulLength--)
	{
		*ulInCRC = (*ulInCRC >> 8) ^ this->ulTable[(*ulInCRC & 0xFF) ^ *sData++];
	}
}
```

CRC_Detect.cpp

```
#include "MainH.h"

CheckSum Check;
void CheckSum::DetectCRC()
{
	Check.CRCInit();
	if (anticheat.CRC_Main == 1)
	{
		long Crc1 = Check.FileCRC(anticheat.ProtectEXE);
		if (Crc1 != anticheat.CRC_MainEXE)
		{
			MessageBoxA(NULL, "파일변조감지 !!", "위험", MB_OK);
		}
	}
}

void MainCRC(void* lpParam)
{
	Check.DetectCRC();
}
```

Detect_SBP.cpp

```
#include "MainH.h"

void SoftwareBreakPoint()
{
	/*int flag = 0;
	__try
	{
		__asm
		{
			int 3;
		}
	}
	__except (EXCEPTION_EXECUTE_HANDLER)
	{
		flag = 1;
	}
	if (flag == 0)
		MessageBox(NULL, "디버깅 감지 !! [int3]", "SBP", MB_OK);*/
}

void SoftwareBreakPoint2()
{
	/*int flag = 0;
	__try
	{
		__asm
		{
			int 0x41
		}
	}
	__except (EXCEPTION_EXECUTE_HANDLER)
	{
		flag = 1;
	}
	if (flag ==0)
		MessageBox(NULL, "디버깅 감지 !! [int41]", "SBP", MB_OK);*/
}

// 윈도우 이름 검색
void CheckSBP()
{
	SoftwareBreakPoint();
	SoftwareBreakPoint2();
}

void SBP_Scan()
{
	CheckSBP();
}

void DetectSBP()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(SBP_Scan), NULL, 0, 0);
}
```

Detect_HBP.cpp

```
#include "MainH.h"

void HardwareBreakPoint()
{
	int flag = 0;
	CONTEXT my;

	ZeroMemory(&my, sizeof(CONTEXT));
	my.ContextFlags = CONTEXT_DEBUG_REGISTERS;

	if (GetThreadContext(GetCurrentThread(), &my) != 0)
	{
		if (my.Dr0 != 0)
			++flag;
		if (my.Dr1 != 0)
			++flag;
		if (my.Dr2 != 0)
			++flag;
		if (my.Dr3 != 0)
			++flag;
	}

	if (flag != 0)
		MessageBox(NULL, "디버깅감지 !! [HBP]", "HBP", MB_OK);
}

void CheckHBP()
{
	HardwareBreakPoint();
}

void HBP_Scan()
{
	CheckHBP();
}

void DetectHBP()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(HBP_Scan), NULL, 0, 0);
}
```

Detect_NtQSI.cpp

```
#include "MainH.h"

typedef struct _SYSTEM_KERNEL_DEBUGGER_INFORMATION
{
	BOOLEAN KdDebuggerEnabled;
	BOOLEAN KdDebuggerNotPresent;
} SYSTEM_KERNEL_DEBUGGER_INFORMATION;

typedef NTSTATUS(WINAPI* ntqsi)(ULONG, PVOID, ULONG, PULONG);

void NtQuerySystem()
{
	SYSTEM_KERNEL_DEBUGGER_INFORMATION sk;
	ntqsi qsi = (ntqsi)GetProcAddress(GetModuleHandle("ntdll.dll"), "NtQuerySystemInformation");
	qsi(0x23, &sk, 2, NULL);
	if (sk.KdDebuggerEnabled && !sk.KdDebuggerNotPresent)
		MessageBox(NULL, "디버깅감지 !!", "NtQSI", MB_OK);
}

void CheckNtQSI()
{
	NtQuerySystem();
}

void NtQSI_Scan()
{
	CheckNtQSI();
}

void DetectNtQSI()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(NtQSI_Scan), NULL, 0, 0);
}
```

Detect_BD.cpp

```
#include "MainH.h"

void BeingDebugged()
{
	if (IsDebuggerPresent())
		MessageBox(NULL, "디버깅감지 !! [BD]", "BD", MB_OK);
}

void CheckBD()
{
	BeingDebugged();
}

void BD_Scan()
{
	CheckBD();
}

void DetectBD()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(BD_Scan), NULL, 0, 0);
}
```

Erase_PE.cpp

```
#include "MainH.h"

void ErasePE()
{
	DWORD pe = 0;
	PCHAR base = (PCHAR)GetModuleHandle(NULL);

	VirtualProtect(base, 4096, PAGE_READWRITE, &pe);
	memset(base, 0, 4096);
}

void PEerase()
{
	ErasePE();
}

void PE_Erase()
{
	PEerase();
}

void AntiPE()
{
	CreateThread(NULL, NULL, LPTHREAD_START_ROUTINE(PE_Erase), NULL, 0, 0);
}
```

Splash.h

```
#ifndef _ABHI_SPLASH_H_
#define _ABHI_SPLASH_H_

#include "windows.h"

class CSplash
{
public:

	CSplash();
	CSplash(LPCTSTR lpszFileName, COLORREF colTrans);
	virtual ~CSplash();
	void ShowSplash();
	int DoLoop();
	int CloseSplash();
	DWORD SetBitmap(LPCTSTR lpszFileName);
	DWORD SetBitmap(HBITMAP hBitmap);
	bool SetTransparentColor(COLORREF col);
	LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
	HWND m_hwnd;

private:
	void Init();
	void  OnPaint(HWND hwnd);
	bool MakeTransparent();
	HWND RegAndCreateWindow();
	COLORREF m_colTrans;
	DWORD m_dwWidth;
	DWORD m_dwHeight;
	void FreeResources();
	HBITMAP m_hBitmap;
	LPCTSTR m_lpszClassName;
};

#endif
```

Splash.cpp.h

```
#include "splash.h"
#include "windowsx.h"

typedef BOOL(WINAPI* lpfnSetLayeredWindowAttributes)
(HWND hWnd, COLORREF cr, BYTE bAlpha, DWORD dwFlags);

lpfnSetLayeredWindowAttributes g_pSetLayeredWindowAttributes;

#define WS_EX_LAYERED 0x00080000

static LRESULT CALLBACK ExtWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	static CSplash* spl = NULL;
	if (uMsg == WM_CREATE)
	{
		spl = (CSplash*)((LPCREATESTRUCT)lParam)->lpCreateParams;
	}
	if (spl)
		return spl->WindowProc(hwnd, uMsg, wParam, lParam);
	else
		return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

LRESULT CALLBACK CSplash::WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	switch (uMsg)
	{
		HANDLE_MSG(hwnd, WM_PAINT, OnPaint);
	}

	return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

void CSplash::OnPaint(HWND hwnd)
{
	if (!m_hBitmap)
		return;

	PAINTSTRUCT ps;
	HDC hDC = BeginPaint(hwnd, &ps);

	RECT   rect;
	::GetClientRect(m_hwnd, &rect);

	HDC hMemDC = ::CreateCompatibleDC(hDC);
	HBITMAP hOldBmp = (HBITMAP)::SelectObject(hMemDC, m_hBitmap);

	BitBlt(hDC, 0, 0, m_dwWidth, m_dwHeight, hMemDC, 0, 0, SRCCOPY);

	::SelectObject(hMemDC, hOldBmp);

	::DeleteDC(hMemDC);

	EndPaint(hwnd, &ps);
}
```

Splash.cpp.h

```
void CSplash::Init()
{
	m_hwnd = NULL;
	m_lpszClassName = TEXT("SPLASH");
	m_colTrans = 0;
	HMODULE hUser32 = GetModuleHandle(TEXT("USER32.DLL"));

	g_pSetLayeredWindowAttributes = (lpfnSetLayeredWindowAttributes)
		GetProcAddress(hUser32, "SetLayeredWindowAttributes");
}

CSplash::CSplash()
{
	Init();
}

CSplash::CSplash(LPCTSTR lpszFileName, COLORREF colTrans)
{
	Init();

	SetBitmap(lpszFileName);
	SetTransparentColor(colTrans);
}

CSplash::~CSplash()
{
	FreeResources();
}

HWND CSplash::RegAndCreateWindow()
{

	WNDCLASSEX wndclass;
	wndclass.cbSize = sizeof(wndclass);
	wndclass.style = CS_BYTEALIGNCLIENT | CS_BYTEALIGNWINDOW;
	wndclass.lpfnWndProc = ExtWndProc;
	wndclass.cbClsExtra = 0;
	wndclass.cbWndExtra = DLGWINDOWEXTRA;
	wndclass.hInstance = ::GetModuleHandle(NULL);
	wndclass.hIcon = NULL;
	wndclass.hCursor = ::LoadCursor(NULL, IDC_WAIT);
	wndclass.hbrBackground = (HBRUSH)::GetStockObject(LTGRAY_BRUSH);
	wndclass.lpszMenuName = NULL;
	wndclass.lpszClassName = m_lpszClassName;
	wndclass.hIconSm = NULL;

	if (!RegisterClassEx(&wndclass))
		return NULL;

	DWORD nScrWidth = ::GetSystemMetrics(SM_CXFULLSCREEN);
	DWORD nScrHeight = ::GetSystemMetrics(SM_CYFULLSCREEN);

	int x = (nScrWidth - m_dwWidth) / 2;
	int y = (nScrHeight - m_dwHeight) / 2;
	m_hwnd = ::CreateWindowEx(WS_EX_TOPMOST | WS_EX_TOOLWINDOW, m_lpszClassName,
		TEXT("Banner"), WS_POPUP, x, y,
		m_dwWidth, m_dwHeight, NULL, NULL, NULL, this);
```

Splash.cpp.h

```
if (m_hwnd)
	{
		MakeTransparent();
		ShowWindow(m_hwnd, SW_SHOW);
		UpdateWindow(m_hwnd);
	}
	return m_hwnd;
}

int CSplash::DoLoop()
{

	if (!m_hwnd)
		ShowSplash();

	MSG msg;
	while (GetMessage(&msg, NULL, 0, 0))
	{
		TranslateMessage(&msg);
		DispatchMessage(&msg);
	}

	return msg.wParam;

}

void CSplash::ShowSplash()
{
	CloseSplash();
	RegAndCreateWindow();
}


DWORD CSplash::SetBitmap(LPCTSTR lpszFileName)
{

	HBITMAP    hBitmap = NULL;
	hBitmap = (HBITMAP)::LoadImage(0, lpszFileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
	return SetBitmap(hBitmap);
}

DWORD CSplash::SetBitmap(HBITMAP hBitmap)
{
	int nRetValue;
	BITMAP  csBitmapSize;
	FreeResources();
	if (hBitmap)
	{
		m_hBitmap = hBitmap;
		nRetValue = ::GetObject(hBitmap, sizeof(csBitmapSize), &csBitmapSize);
		if (nRetValue == 0)
		{
			FreeResources();
			return 0;
		}
		m_dwWidth = (DWORD)csBitmapSize.bmWidth;
		m_dwHeight = (DWORD)csBitmapSize.bmHeight;
	}
	return 1;
}
```

Splash.cpp

```
void CSplash::FreeResources()
{
	if (m_hBitmap)
		::DeleteObject(m_hBitmap);
	m_hBitmap = NULL;
}

int CSplash::CloseSplash()
{

	if (m_hwnd)
	{
		DestroyWindow(m_hwnd);
		m_hwnd = 0;
		UnregisterClass(m_lpszClassName, ::GetModuleHandle(NULL));
		return 1;
	}
	return 0;
}

bool CSplash::SetTransparentColor(COLORREF col)
{
	m_colTrans = col;

	return MakeTransparent();
}

bool CSplash::MakeTransparent()
{
	if (m_hwnd && g_pSetLayeredWindowAttributes && m_colTrans)
	{
		SetWindowLong(m_hwnd, GWL_EXSTYLE, GetWindowLong(m_hwnd, GWL_EXSTYLE) | WS_EX_LAYERED);
		g_pSetLayeredWindowAttributes(m_hwnd, m_colTrans, 0, LWA_COLORKEY);
	}
	return TRUE;
}
```

Splash2.cpp

```
#include "Splash.h"


void SplashShow() {
	CSplash splash1(TEXT("AntiCheat.bmp"), RGB(128, 128, 128));
	splash1.ShowSplash();
	Sleep(2000);
	splash1.CloseSplash();
}
```