DirectX를 이용한 월 핵(WALL HACK) 제작 과정

지도교수 : 한 상 훈
연 구 자 : 김 민 석

< 목 차 >

요 약

게임 시장이 점점 증가하는 추세인 요즘, 악성 게임 유저인 핵 유저들 또한 증가 추세이다, 이러한 현상 속에서 게임 핵 또는 비인가 프로그램으로 불리는 흔히 핵 이라는 주제를 가지고 게임 핵 제작에 대한 오해와 그와 연관된 방어 방법, 처벌과 핵의 종류를 알아본다. 따라서 게임 핵 제작에 대해 알아보기 위하여 D3D9 그래픽 라이브러리를 활용하여 간단한 게임 핵을 제작해본다. Z Buffer의 원리에 대해 알아보고 그걸 위한 후킹 과정(DLL인젝션 -> vtable 주소 찾기-> DrawIndexedPrimitive 함수 주소 구하기 -> Inline Hook)을 알아본다.

주요어 : 게임 핵, D3D9, Inline Hook

1. 서론

1.1 게임 핵이란

게임 내 해킹 프로그램을 이르는 말이며 흔히 핵이라고 불리며, 여러 게임 내 텍스트에서는 이를 "서드 파티 프로그램", 혹은 "비인가 프로그램"이라고 지칭한다. 일부 개발자들이 "불법 프로그램"이라고도 지칭하지만 프로그램의 '유포'가 불법일 뿐 프로그램 자체는 불법이 아니다. 프로그램 혹은 게임 내의 스크립트을 해킹하여 그것을 수정하여 원래 동작과 다른 동작을 하도록 유도하는 용도로 주로 사용한다. 흔히 말하는 맵 핵, 스피드 핵, 에임 봇 등이 해킹 프로그램의 범주에 포함된다. 국내에서는 게임산업진흥에 관한 법률에 따라 유포할 경우 불법이다. 국내에서도 불특정 다수에게 “유포”시만 처벌되고 단순 이용자는 처벌되지 않는다. 전 세계 대부분의 다른 국가에서는 관련법이 없기 때문에 만들어진 핵을 상업용으로 판매하는 등 저작권법에 저촉되는 부분이 없다면 합법적인 것으로 본다. 하지만 쓰지 않는 것이 그 게임과 게임을 플레이 하는 유저를 위하는 길이다.

원 제작자 측에선 막으려고 하는 경우가 대다수이지만 드물게 제작자 측에서 모드 프로그램으로 인정하는 경우도 있다. 와우(WOW, World Of Warcraft)의 애드온(Addon)이나 월드 오브 탱크(World Of Tanks) PC 버전이 대표적인 사례. 하스스톤(HearthStone)의 덱트래커(Deck Tracker) 또한 제작진이 공식적으로 인정한 프로그램이다. 위 경우에는 사용해도 문제가 없다. 당연하지만 어디까지 편의를 봐주는 프로그램으로, 에임 핵이나 월 핵처럼 게임에 직접적 영향을 미치는 프로그램이 인정된 경우는 지금까지 단 한 번도 없었다.

핵 개발에 대부분 사용하는 프로그래밍 언어는 메모리를 수정하는 경우 포인터에 접근할 수 있는 C++과 C#, 간단한 매크로인 경우는 Lua와 AutoHotKey등이 사용되며, 웹 게임에 사용하는 핵은 JavaScript로 만든다.

핵이 유포가 될 경우 게임 화폐 시세가 하락하는 경우도 있으며 일반 유저들에게 많은 피해를 주게 된다. 그리고 이 상황이 계속 진행될 경우 결국 유저들은 그 게임을 떠나게 되며 최악의 경우에는 게임 자체가 서비스 종료되는 상황을 초래하기도 한다.

2. 연구과정

2.1 게임 핵 제작에 대한 오해

게임 핵 제작을 어려운 기술로 보는 경향이 있지만 정확한 것은 아니다. 물론 현대 안티 치트와 보안 프로그램으로 보호된 데이터를 도움 없이 스스로 다 파헤쳐 내는 것은 어려운 것이 맞다. 하지만 보안 프로그램이 없는 싱글플레이어 게임이나 보안 프로그램의 우회법이 널리 알려진 게임들의 경우 프로그래밍에 대한 기초가 있고 게임프로그램 설계에 대해 관심이 많다면, 이미 구현된 툴을 이용하여 쉽게 만들 수 있다. 그나마 해킹 관련 전문적인 지식이 필요한 부분들은 관련 툴 없이 후킹하는 방법과 리버싱, 패킷변환 등이 있는데, 패킷변조 외에는 오래된 방법이고 이미 치트 엔진과 같은 GUI 툴이 쉽게 제공하고 있어서 사라진 방법들이다. 패킷변조도 패킷이 관여하는 특정 기술 외에는 불필요하다. 이는 즉, 특정기술들을 제외하면 기본기술들은 전문지식이 없어도 게임 핵 제작이 가능함을 말한다. 그래서 한국에서조차도 청소년들이 비인가 프로그램(게임 핵)을 만들어 유포하다 적발되는 사례가 적지 않다.

그럼에도 불구하고 대중에게는 핵 제작자들이 해커로서 고급 인재라고 생각되는 경우도 있는데, 실상은 아니다. 이유는 다음과 같다.

핵 제작 후 유포는 범죄 행위이다. 특히나 유포가 적발될 경우 전과가 남기도 한다. 범죄 조직이 아닌 이상 어떤 조직도 전과자를 선호 하지는 않을 것이다. 전과와 무관하게, 이런 사람들은 회사입장에서 이들을 좋은 시선으로 보지 않는다. 이런 사람들은 회사 내 프로그램이나 시스템을 악용할 수 있다고 생각하기 때문에 기피하는 것이다. 어떤 회사도 몰래 백도어[1]를 심을 수 있는 사람과 일을 하려고 하지 않는다. 이는 반대로 대부분의 프로그래머들이 게임 핵을 만들 줄 몰라서 안하는 경우가 아니라 불법 행위이기 때문에 하지 않는다. 정상적인 취업활동을 하는 경우 핵을 제작 하였더라도 자신의 이력서에 핵 제작 이력을 당당히 적지 않는다.

게임 핵은 정보보안에서는 매우 낮은 수준으로 취급된다. 특히 패킷 변조조차 없는 핵이라면 메모리 값 변조밖에 되지 않기에 뛰어난 실력을 가졌다고 보기 어렵다. 패킷변조를 하였더라도 패킷 변조에 대한 방어를 제공하는 안티 치트 솔루션을 도입하면 될 정도로 게임 핵 제작은 경력으로서 무의미한 경력이다. 단순히 핵과 패킷변조를 만들 수 있다고 게임의 보안 알고리즘 및 수단과 방법까지 알고 있다는 것이 아니기 때문이다. 게다가 대부분의 게임의 경우 취약점이 중대한 문제가 아니라면 고치는 것보다는 새로운 컨텐츠에 전념하기에 일부러 놔두는 경향도 있어서 취약점을 알고 있다는 것에 큰 이점도 없다.

잘못 알려진 정보 중에 깃 허브(GitHub)에 특정 게임의 핵 소스 코드가 공개되어 이를 바탕으로 게임 핵이 만들어진다고 알려져 있지만, 사실과는 다르다. 게임사나 게임 개발자가 핵사용자들로부터 수입을 유지하기 위함으로 비공식적으로 일부러 공개하지 않는 이상, 오픈 소스로 공개되는 경우는 매우 드물다. 오히려 소스코드가 아닌 온라인이 필요한 코드를 DLL형태로 브로커 들에게 판매하고 있는 경우가 대부분이다. 브로커 들은 그 DLL 핵 코어를 토대로 가공하여 게임 핵을 제작하여 시중에 유포하거나 판매한다. 게다가 오픈 소스로의 공개는 당연히 게임사에게도 손쉽게 취약점이 알려지기 때문에 대응을 빠르게 할 수 있다는 점을 간과한 루머에 불과하다.

반대로 오픈소스로 공개되는 부분이 있기는 하지만, 핵 소스코드라 할 정도로 특정 게임에 맞춘 핵 프로그램을 말하는 것은 아니고 핵을 쉽게 만들 수 있게 해주는 치트 엔진의 안티 치트 별 바이패스 버전이다. 치트 엔진 자체가 오픈 소스이기에 안티 치트에 대한 편법들을 적용한 치트 엔진의 소스코드가 매우 많이 만들어진다. 게임에 대한 적용은 따로 해줘야하기 때문에 이 오픈 소스만으로 특정 게임의 핵이라고는 할 수 없지만, 오픈 소스 기반으로 게임 핵이 만들어지는 것은 사실이다.

핵을 제작하여 유포하다 적발되는 경우 청소년들이나 학생들이 있으면, 이들을 인재로 보며 여러 통신매체에서 보도하기 때문에, 일반인들은 핵 제작자들 대부분이 청소년들이나 중고등학생들이라 생각하기도 한다. 그렇지만 정작 핵심적인 개발자들은 모두 성인들로 대학생 및 대학원생이나 직장인 인 경우가 대다수다. 적발하기 쉬운 통상적인 핵들의 경우에는 프로그래밍 기술들에 관심 있는 누구나 쉽게 만들 수 있지만, 핵을 사용하다 들키는 것을 최소화하기 위한 수단으로 우회와 위장 등의 전문적인 기술이 필요한 핵의 경우 대부분 대학 수준 이상의 지식이 필요하다. 그래서 기존에 안티 치트를 높은 수준으로 도입한 게임들의 핵들은 청소년들이나 중고등학생들만으로 제작 할 수 없으며, 만약 이들이 제작하였다면

1) 하드웨어나 소프트웨어 등의 개발과정이나 유통과정 중에 몰래 탑재되어 정상적인 인증 과정을 거치지 않고 보안을 해제할 수 있도록 만드는 악성코드이다.

즉시 적발될 정도이다. 그나마 이러한 상황에서도 유지되는 핵들은 게임사가 비공식 허용했거나 핵제 작에 대해 개발할 시간이 있는 대학생 및 대학원생이나 직장인이 제작한 경우라고 볼 수 있다.

2.2 방어

운영자들은 어느 정도 핵을 막기 위한 여러 가지 방법을 사용하고 있다. 안티 치트를 이용하여 핵 유저들을 잡아내는 것과 클라이언트 변조 방지를 위해 클라이언트를 패킹하여 변조하기 어렵게 만드는 것이다.

가짜 핵을 만들어 시중에 배포해 핵 사용자를 적발하거나, 일반적인 핵이더라도 사용자도 모르게 자진신고를 하는 기능을 넣어 계정을 쉽게 찾아내도록 하는 등 여러 방법이 있다.

2.3 처벌

한국에서는 게임 핵을 만들어 팔거나 유포하는 것은 불법이다. 한국을 제외한 다른 나라에서도 개발자들이 게임 핵을 잡으려 고는 하나 현실적으로는 불가능하다. 해커가 IP를 고정해놓고 활동하는 게 아니라 항상 IP를 바꾸면서 익명으로 활동한다. 또한 지나치게 많은 핵 유저들도 문제이며 핵 유저를 잡는 경찰의 인력 부족이 현실이다. 제일 중요한 것이 핵 유저들의 뿌리인 해커를 잡아내는 것이 매우 힘들다는 것과 당장 몇 년을 추적해도 못 잡는 경우가 많다.

그러므로 중요한 것은 게임사의 행동에 달렸다. 핵 유저를 잡은 즉시 처벌하거나 하면 어느 정도는 해결될 수 는 있지만 오로지 안티 치트에만 의존한다면 해결되지 않는다. 게임을 운영하는 것은 사람이지 안티 치트가 아니다.

2.4 핵 종류

스피드 핵: 캐릭터/차량의 속도를 비정상적으로 증가/감소시킨다.

오브젝트 핵: 폭파되어 수동으로 조종할 수 없는 차량 등의 게임 내 이동 불가능 개체를 강제로 이동시킨다.

ESP: 핵 사용자의 카메라 안에 들어오는 플레이어들의 위치를 보이게 한다.

에임 핵: 상대의 특정 부위(예: 머리)에 조준점이 고정되며 반동이 제거되거나 감소한다.

탄속 핵: FPS게임 같이 탄속이 있는 게임에서 탄속을 더 빠르게 또는 더 느리게 만든다.

맵 핵: ESP와 비슷한 역할이지만, RTS 류 게임에서 보이지 않는 위치의 적을 보여주는 핵.

고스트 핵: 적의 공격에 피해를 입지 않거나 적에게 보이지 않게 한다.

블랙홀 핵: 상대를 특정 지점으로 강제로 끌고 오거나 강제로 텔레포트 시킨다.

대미지/쿨감 핵: 이름 그대로 대미지 증가/데미지 감소, 쿨 다운이 있는 행위의 쿨 타임을 제거하거나 줄인다.

관통(노클립) 핵: 일부 FPS 게임에서는 벽뚫이나 노클립이라고도 불리며 날아다니며 충돌 판정이 있는 오브젝트나 지형을 무시하고 통과할 수 있다.

무적 핵: 자신을 무적으로 만들거나 HP를 무한으로 늘린다.

아이템 핵: 모든 아이템을 즉시 소지 상태로 만들거나 더미 데이터 등 일반적으로 얻을 수 없는 아이템을 가져온다.

자원 핵: 게임 내의 화폐의 수치를 조작하여 늘리거나 줄일 수 있다.

팀킬 핵: 팀킬이 허용되지 않는 일부 FPS 게임에서 같은 편에게 데미지를 입히거나 죽일 수 있다.

킬올 핵: 자신 또는 자신이 예외로 설정한 인원을 제외한 서버 또는 매치 내의 모든 인원을 즉시 죽인다.

관리자 핵: 관리자 이상의 권한을 가져온다.

플라잉 핵: 비행을 가능하게 해주는 핵으로, 노클립과 비슷하지만 지형지물 관통보다는 공중부양에 중점을 둔 핵이다.

3. 제작과정

FPS 게임에서 주로 발견되는 Wall Hack (월 핵)은 말 그대로 벽 뒤에 있는 적의 위치 정보를 보여준다. 월 핵을 만드는 방법은 그래픽 렌더링 라이브러리마다 조금씩 다르다.

[사진] 1 월 핵 적용 중인 게임화면

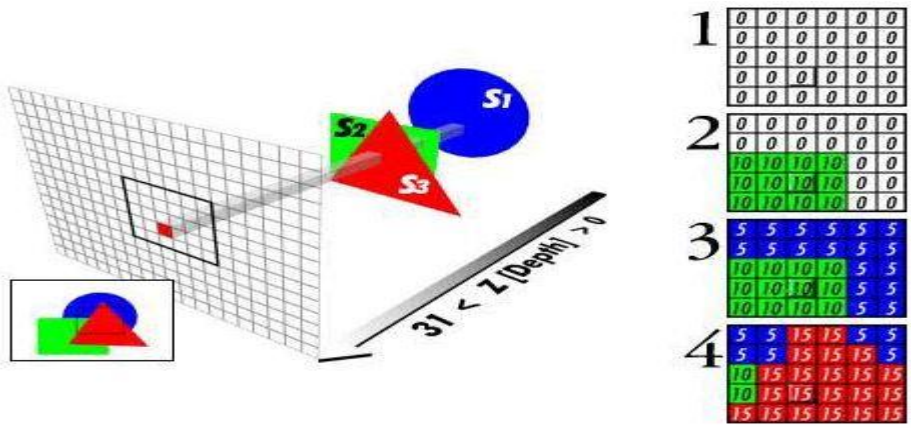
그래픽 렌더링 라이브러리 중 Microsoft에서 개발한 Direct X(Direct3D)로 개발된 게임의 월 핵을 구현하는 방법을 소개하도록 한다.

- Z Buffer (Depth Buffer)

D3D9에서 월 핵을 구현하기 위해선 렌더링에서 사용되는 Z Buffer의 개념을 알아야 한다. Z Buffer란 렌더링할 때 어떤 물체가 보여야 할지에 대한 여부를 판단하기 위해 사용되는 방법 중 하나다. Z Buffer는 아래와 같다.

어떤 물체가 그려질 때 만들어진 픽셀의 깊이 정보(z 좌표)는 버퍼(Z 버퍼 혹은 깊이 버퍼)에 저장된다. 이 버퍼는 (x-y)의 2차원 좌표를 기준으로 해당하는 각각의 스크린 픽셀 요소들로 정렬되어 있다. 만약 다른 물체가 같은 픽셀에 그려져야 할 때, Z 버퍼링은 현재 픽셀과 새로 그려질 픽셀 중 어떤 것이 관찰자에게 더 가까운지 깊이를 비교한다. Z 버퍼에 기록되도록 새로 선택된 깊이는 이전의 깊이를 덮어쓴다. 즉, Z 버퍼는 '더 가까운 물체가 더 먼 물체를 가린다'라는 깊이 관념을 정확하게 따를 수 있게 돕는다.

아래 그림처럼 3개의 각각의 도형이 있는 상황에 Z Buffer가 어떤 방식으로 작동하는지 알아보자.



[사진 2] Z Buffer 동작

이 그림에서는 도형을 S2, S1, S3의 순으로 그리고 있다.

오른쪽에 있는 그림은 2차원 배열이 초기화된 직후의 Z Buffer 상태부터 S2, S1, S3 도형을 순서대로 그린 후의 Z Buffer 상태이다. 각각의 상태에 대해 해석해보자.

1. 아무것도 그리지 않은 초기 Z Buffer 상태이다.

2. S2 도형을 그리고 난 후 Z Buffer 상태이다.

3. S1 도형을 그리고 난 후 Z Buffer 상태이다. S2 도형과 겹치는 부분은 깊이 값을 비교했을 때 Z Buffer에 기록되어 있는 값이 더 크기 때문에 해당 영역은 업데이트 되지 않았다.

4. S3 도형을 그리고 난 후 Z Buffer 상태이다. S1, S2 도형과 겹치는 부분은 깊이 값을 비교했을 때 Z Buffer에 기록되어 있는 값이 더 작기 때문에 해당 영역은 S3 도형의 깊이 값으로 업데이트 되었다.

만약 특정 물체(적 플레이어)를 렌더링할 때 Z Buffer 기능을 비활성화한다면 렌더링 엔진은 해당 물체가 보여야 할지의 여부를 구별할 수 없기 때문에 해당하는 물체를 항상 화면에 보여줄 것이다. 이것이 Z Buffer를 사용한 월 핵의 기본적인 원리이다. 다만 Z Buffer 기능을 비활성화하기 위해선 렌더링의 흐름을 제어해야 가능한데 이를 위해서는 Direct3D의 라이브러리 함수를 후킹해야 한다.

- D3D9 Hook

D3D9의 함수를 후킹을 진행하려면 후킹 할 함수의 주소를 알아내는 것이 필요로 한다. 먼저 Direct3D가 작동하는 기본적인 원리를 알아보자.

아래는 Direct3D 인터페이스를 생성하는 코드이다.

```
1  // this function initializes and prepares Direct3D for use
2  void initD3D(HWND hWnd){
3      d3d = Direct3DCreate9(D3D_SDK_VERSION);    // create the Direct3D interface
4
5      D3DPRESENT_PARAMETERS d3dpp;    // create a struct to hold various device information
6
7      ZeroMemory(&d3dpp, sizeof(d3dpp));    // clear out the struct for use
8      d3dpp.Windowed = TRUE;    // program windowed, not fullscreen
9      d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;    // discard old frames
10     d3dpp.hDeviceWindow = hWnd;    // set the window to be used by Direct3D
11
12     // create a device class using this information and information from the d3dpp stuct
13     d3d->CreateDevice(D3DADAPTER_DEFAULT,
14                       D3DDEVTYPE_HAL,
15                       hWnd,
16                       D3DCREATE_SOFTWARE_VERTEXPROCESSING,
17                       &d3dpp,
18                       &d3ddev);
19 }
```

[사진 3] IDirect3DDevice9 인터페이스 호출

Direct3D 라이브러리는 대부분의 export 되어 있는 형태가 아니기 때문에 CreateDevice 함수를 통해 생성된 IDirect3DDevice9 인터페이스를 사용하여 렌더링 함수들을 호출해야 한다.

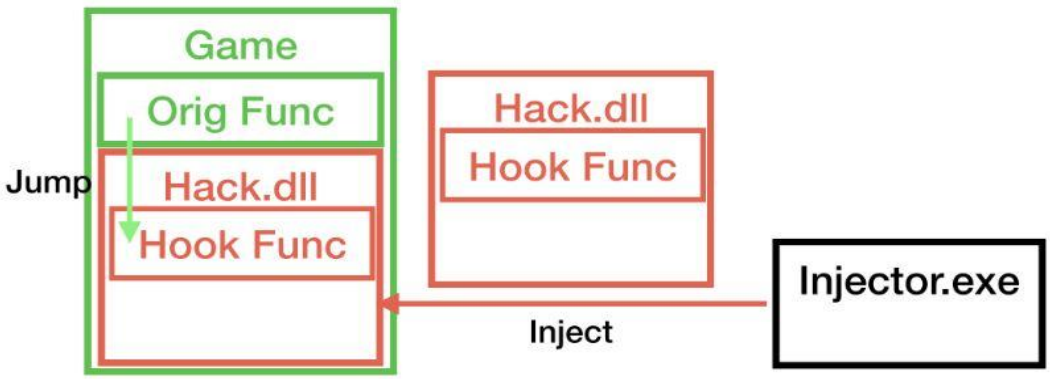
이 정보들을 바탕으로 D3D9의 함수를 후킹하는 과정을 요약하면 다음과 같다.

1. 게임 프로세스의 메모리에 접근하기 위해 DLL을 인젝션한다.
2. vtable의 주소를 찾는다.
3. DrawIndexedPrimitive 함수의 주소를 구한다.
4. 함수에 Inline Hook을 설치한다.
5. 숨겨진 물체를 보이게 한다.

여기서 vtable이란 C++ 클래스의 멤버 함수 중 가상 함수가 존재하는 경우 생성되는 함수 포인터 배열이다. 따라서 vtable 주소만 구하면 모든 가상 함수의 주소를 찾을 수 있게 된다.

3.1 DLL 인젝션

DLL Injection 없이 외부 프로세스에서 메모리 관련 WIN API를 사용하는 방식(External Hook이라고도 함)으로도 월 핵 구현이 가능하지만 보통 개발이 편한 DLL Injection을 사용한다. 그림으로 간략하게 표현 하겠다.



[사진 4] DLL Injection

3.2 Vtable 주소 찾기

이제 IDirect3DDevice9 인터페이스의 vtable을 찾아야 한다. 이 인터페이스는 CreateDevice 함수를 통해서 생성되기 때문에 이 함수를 중심으로 분석해야 한다.

CreateDevice 함수는 7번째 인자인 ppReturnedDeviceInterface로 IDirect3DDevice9 인터페이스를 반환한다. ppReturnedDeviceInterface 변수는 CreateDevice 함수 시작 시 0으로 초기화된 후 다음 부분에서 설정된다.

```
LONG __stdcall CEnum::CreateDevice(
        CEnum *this, // this 포인터가 때문에 msdn에는 이 부분이 없음
        unsigned int a2,
        enum _D3DDEVTYPE a3,
        HWND a4,
        unsigned int a5,
        struct _D3DPRESENT_PARAMETERS_ *a6,
        struct IDirect3DDevice9 **ppReturnedDeviceInterface)
{
/*
HRESULT CreateDevice(
  UINT                    Adapter,
  D3DDEVTYPE              DeviceType,
  HWND                    hFocusWindow,
  DWORD                   BehaviorFlags,
  D3DPRESENT_PARAMETERS *pPresentationParameters,
  IDirect3DDevice9       **ppReturnedDeviceInterface
);
*/
...
v12 = CEnum::CreateDeviceImpl(
            (CEnum *)v7,
            v8,
            a3,
            (HWND)*(&var_164 + 1),
            a5,
            v23,
            v22,
            (struct IDirect3DDevice9Ex **)&var_164,
            v11);
    v13 = var_164;
    *(&var_164 + 1) = v12;
    *ppReturnedDeviceInterface = (struct IDirect3DDevice9 *)var_164; // 이 곳에서 인터페이스 값이 설정됨
...
```

[사진 5] CreateDevice 함수

var_164 값으로 ppReturnedDeviceInterface 값이 설정되는데 이 값은 CEnum::CreateDeviceImpl 함수의 8번째 인자에서 설정된다. CreateDeviceImpl 함수를 따라가 보면 다음과 같다.

```
int __thiscall CEnum::CreateDeviceImpl(
        CEnum *this,
        unsigned int a2,
        enum _D3DDEVTYPE a3,
        HWND a4,
        unsigned int a5,
        struct _D3DPRESENT_PARAMETERS_ *arg_10,
        const struct D3DDISPLAYMODEEX *a7,
        struct IDirect3DDevice9Ex **ppReturnedDeviceInterface,
        struct _D3D9ON12_ARGS *a9
){
    if ( hLibModule )
    {
      v26 = CD3DHal::CD3DHal((CD3DHal *)hLibModule); // v26은 이 함수의 반환값으로 설정됨
      goto LABEL_36;
    }
...
    if ( !v27 )
    {
      *ppReturnedDeviceInterface = (struct IDirect3DDevice9Ex *)v26; // ppReturnedDeviceInterface에
      return 0;
    }
    (*(void (__thiscall **)(CD3DHal *, int))(*(_DWORD *)v26 + 696))(v26, 1);
    D3DRecordHRESULT(
      (size_t)"Failed to initialize D3DDevice. CreateDeviceEx Failed.",
      (struct _hrCapture *)0xDEADBEEF,
      "windows₩₩directx₩₩dxg₩₩inactive₩₩d3d9₩₩d3d₩₩fe₩₩d3ddev.cpp",
      1068);
```

[사진 6] CreateDeviceImpl 함수

ppReturnedDeviceInterface 에 들어가는 값은 v26에서 왔고, v26은 CD3DHal::CD3DHal 함수에서 왔다. CD3DHal::CD3DHal 함수는 아래와 같다.

```
1  CD3DHal *__thiscall CD3DHal::CD3DHal(CD3DHal *this){
2    CD3DHal *v1; // esi
3
4    v1 = this;
5    CD3DBase::CD3DBase(this);
6    *(_DWORD *)v1 = &CD3DHal::`vftable'; // vtable을 초기화 하는 부분
7    *((_DWORD *)v1 + 3220) = 0;
8    *((_DWORD *)v1 + 3218) = 0;
9    *((_DWORD *)v1 + 3219) = 0;
10   *((_DWORD *)v1 + 3269) = 0;
11   *((_DWORD *)v1 + 3272) = 0;
12   *((_DWORD *)v1 + 3278) = 0;
13   *((_DWORD *)v1 + 3279) = 0;
14   *((_DWORD *)v1 + 3280) = 0;
15   *((_DWORD *)v1 + 3281) = 0;
16   *((_DWORD *)v1 + 4088) = 0;
17   *((_DWORD *)v1 + 4091) = 0;
18   *((_DWORD *)v1 + 4094) = 0;
19   *((_DWORD *)v1 + 4097) = 0;
20   *((_DWORD *)v1 + 4100) = 0;
21   *((_BYTE *)v1 + 16404) = 0;
22   *((_DWORD *)v1 + 3275) = 0;
23   return v1;
24 }
```

[사진 7] CD3DHal::CD3DHal 함수

6번째 라인에 보이는 &CD3DHal::`vftable'이 IDirect3DDevice9 인터페이스의 vtable이다.

이제 CD3DHal::CD3DHal 함수의 코드 부분을 패턴으로써 사용하여 d3d9.dll 상의 메모리를 스캔해 이 함수를 찾고 vtable의 주소를 알아낼 수 있다.

패턴을 찾기 위해 어셈블리어로 확인해보면 다음과 같다.

```
public: __thiscall CD3DHal::CD3DHal(void) proc near
8B FF             mov     edi, edi
56                push    esi
8B F1             mov     esi, ecx
E8 05 73 00 00    call    CD3DBase::CD3DBase(void)
33 C0             xor     eax, eax
C7 06 24 1D 00 10 mov     dword ptr [esi], offset const CD3DHal::`vftable' // 이 부분부터 패턴 시작
89 86 50 32 00 00 mov     [esi+3250h], eax
89 86 48 32 00 00 mov     [esi+3248h], eax
89 86 4C 32 00 00 mov     [esi+324Ch], eax
89 86 14 33 00 00 mov     [esi+3314h], eax
89 86 20 33 00 00 mov     [esi+3320h], eax
89 86 38 33 00 00 mov     [esi+3338h], eax
89 86 3C 33 00 00 mov     [esi+333Ch], eax
89 86 40 33 00 00 mov     [esi+3340h], eax
89 86 44 33 00 00 mov     [esi+3344h], eax
89 86 E0 3F 00 00 mov     [esi+3FE0h], eax
89 86 EC 3F 00 00 mov     [esi+3FECh], eax
89 86 F8 3F 00 00 mov     [esi+3FF8h], eax
89 86 04 40 00 00 mov     [esi+4004h], eax
89 86 10 40 00 00 mov     [esi+4010h], eax
88 86 14 40 00 00 mov     [esi+4014h], al
89 86 2C 33 00 00 mov     [esi+332Ch], eax
8B C6             mov     eax, esi
5E                pop     esi
C3                retn
                  public: __thiscall CD3DHal::CD3DHal(void) endp
```

[사진 8] CD3DHal::CD3DHal 함수 어셈블리어

vtable을 설정하는 부분부터 추출한 옵코드는 C7 06 24 1D 00 10 89 86 50 32 00 00 89 86이다. 따라서 가변적인 부분을 ??로 치환하면 C7 06 ?? ?? ?? ?? 89 86 ?? ?? ?? ?? 89 86 가 되고, 이것이 최종적으로 vtable을 찾을 때 사용하게 될 패턴이다.

패턴을 찾을 때는 아래 형태의 함수를 많이 사용한다.

```
bool bCompare(const BYTE* pData, const BYTE* bMask, const char* szMask){
    for(;*szMask;++szMask,++pData,++bMask)
        if(*szMask=='x' && *pData!=*bMask )
            return false;

    return (*szMask) == NULL;
}

DWORD FindPattern(DWORD dwAddress,DWORD dwLen,BYTE *bMask,char * szMask){
    for(DWORD i=0; i < dwLen; i++)
        if( bCompare( (BYTE*)( dwAddress+i ),bMask,szMask) )
            return (DWORD)(dwAddress+i);

    return 0;
}
```

[사진 9] FindPattern 함수

구한 패턴과 FindPattern 함수를 사용하여 다음과 같이 vtable의 주소를 구할 수 있다.

```
DWORD table = FindPattern(
    (DWORD)hModule,
    0x128000,
    (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86",
    "xx????xx????xx" // 가변적인 주소 부분은 ?로 마스킹
);
memcpy(&vTable, (void*)(table+2), 4);  // vtable 주소 값을 복사
```

[사진 10] vtable 주소

3.3 DrawIndexedPrimitive 함수 주소를 구하기

D3D9에서 물체 혹은 도형을 그릴 때 사용하는 함수로는 DrawIndexedPrimitive, DrawIndexedPrimitiveUp, DrawPrimitiveUp, DrawPrimitive 등이 있다. 이 중 DrawIndexedPrimitive 함수를 사용한다.

DrawIndexedPrimitive (이하 'DIP') 함수 주소를 찾을 때는 vtable index를 이용한다. 컴파일된 d3d9.dll 바이너리는 vtable index가 고정되어 있기 때문에 미리 구한 vtable index를 이용할 수 있다.

```
#define QUERY_INTERFACE             0
#define ADDREF                      1
#define RELEASE                     2
#define TESTCOOPERATIVELEVEL        3
#define GETAVAILABLETEXTUREMEM      4
...
#define ENDSCENE                    42
...
#define DRAWINDEXEDPRIMITIVE    82
#define DRAWPRIMITIVEUP         83
#define DRAWINDEXEDPRIMITIVEUP  84
```

[사진 11] IDA

IDA를 통해 DRAWINDEXEDPRIMITIVE의 vtable index가 맞는지 확인해보자.

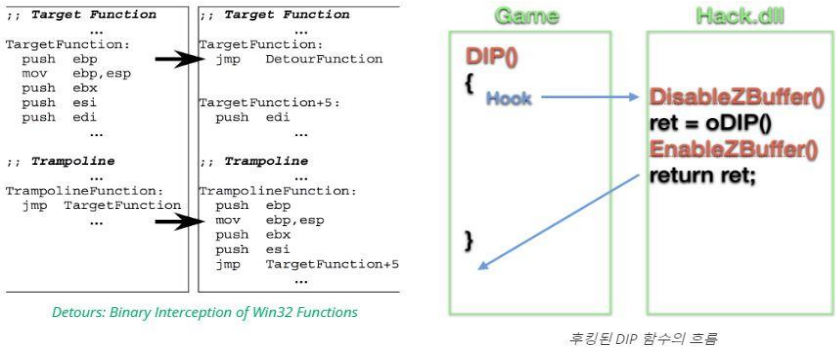
```
.text:10001D24 const CD3DHal::`vftable' dd offset CBaseDevice::QueryInterface(_GUID const &,void * *)
...
.text:10001E6C                          dd offset CD3DBase::DrawIndexedPrimitive(_D3DPRIMITIVETYPE,int,
```

[사진 12] vtable의 주소

vtable의 주소는 0x10001D24 이고 DRAWINDEXEDPRIMITIVE값은 82 이므로
0x10001D24 + 82 * 4(32-bit 포인터 크기) == 0x10001E6C
CD3DBase::DrawIndexedPrimitive 함수 포인터의 주소 값과 동일한 걸 확인할 수 있다.

3.4 함수에 Inline Hook을 설치하기

DIP 함수의 주소를 구했으니 이제 함수를 후킹해 Z Buffer 기능을 비활성화해야 한다. Inline Hook 기법에 대해 설명한다. 먼저 Inline Hook의 원리만 간단히 짚고 넘어가도록 한다.



[사진 13] Inline Hook

좌측 그림을 보면 TargetFunction(후킹 대상 함수, 우측 그림의 DIP 함수에 해당)의 프롤로그를 jmp 명령어로 패치하여 후킹 함수 (사진에서는 DetourFunction)을 실행하도록 한다. Trampoline은 후킹했던 함수의 원본 함수를 호출하고 싶을 때 사용하면 된다(우측 oDIP 함수에 해당). Trampoline 코드에 후킹으로 인해 유실되었던 프롤로그를 복사 후 Original Function+ 51로 점프하게 해 원본 함수을 사용 가능하게 한다.

간단하게 “hooked!₩n”를 출력하는 함수로 후킹 하면 다음과 같은 형태가 된다.

```
#define DRAWINDEXEDPRIMITIVE    82 // DrawIndexedPrimitive의 vtable index

DWORD FindPattern(DWORD dwAddress, DWORD dwLen, BYTE *bMask, char * szMask){
        // 생략
} // 2. 에서 언급한 FindPattern 함수

typedef HRESULT(WINAPI* tDrawIndexedPrimitive)(
        LPDIRECT3DDEVICE9 pDevice,
        D3DPRIMITIVETYPE pType,
        INT BaseVertexIndex,
        UINT MinVertexIndex,
        UINT NumVertices,
        UINT startIndex,
        UINT primCount
); // DrawIndexedPrimitive 함수 원형 선언
tDrawIndexedPrimitive oDrawIndexedPrimitive = NULL;

void *DetourFunction(BYTE *src, const BYTE *dst, const int len){ // inline hook을 설치하는 함수
        BYTE *jmp = (BYTE*)malloc(len + 5);
        DWORD dwBack;

        VirtualProtect(src, len, PAGE_EXECUTE_READWRITE, &dwBack);
        memcpy(jmp, src, len);
        jmp += len;
        jmp[0] = 0xE9;
        *(DWORD*)(jmp + 1) = (DWORD)(src + len - jmp) - 5;
        src[0] = 0xE9;
        *(DWORD*)(src + 1) = (DWORD)(dst - src) - 5;
        for (int i = 5; i < len; i++)
                src[i] = 0x90;
        VirtualProtect(src, len, dwBack, &dwBack);
        VirtualProtect(jmp, len, PAGE_EXECUTE_READWRITE, &dwBack);
        return (jmp - len);
}
```

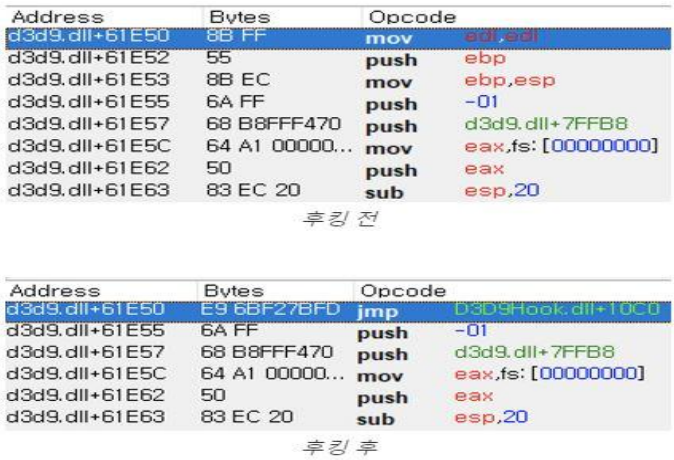
[사진 14] hooked!\n"를 출력하는 함수로 후킹1

```
36  HRESULT __stdcall hkDrawIndexedPrimitive(
37          LPDIRECT3DDEVICE9 pDevice,
38          D3DPRIMITIVETYPE pType,
39          INT BaseVertexIndex,
40          UINT MinVertexIndex,
41          UINT NumVertices,
42          UINT startIndex,
43          UINT primCount
44  ){
45          // "hooked!₩n"를 출력하는 함수
46          printf("hooked!₩n");
47          return oDrawIndexedPrimitive(pDevice, pType, BaseVertexIndex, MinVertexIndex, NumVertices, sta
48  }
49
50  void initHook(void) {
51          DWORD hD3D = (DWORD)GetModuleHandle(L"d3d9.dll");
52          DWORD addr = FindPattern(
53                  hD3D,
54                  0x128000, // d3d9.dll 모듈 크기, 버전별로 다를 수 있음
55                  (PBYTE)"₩xC7₩x06₩x00₩x00₩x00₩x00₩x89₩x86₩x00₩x00₩x00₩x00₩x89₩x86",
56                  "xx????xx????xx"
57          );
58          if (addr) {
59                  DWORD vtableAddress;
60                  memcpy(&vtableAddress, (void *)(addr + 2), 4);
61                  oDrawIndexedPrimitive = (tDrawIndexedPrimitive)DetourFunction(
62                          (PBYTE)vtableAddress[DRAWINDEXEDPRIMITIVE],
63                          (PBYTE)hkDrawIndexedPrimitive,
64                          5
65                  );
66          }
67  }
```

[사진 15] hooked!\n"를 출력하는 함수로 후킹2

주의해야 할 사항은 x86에서 DrawIndexedPrimitive 함수는 호출 규약이 호출된 함수가 스택을 정리하는 stdcall 이라는 점이다. 후킹 시 호출규약이 다르다면 스택 오프셋이 달라지기 때문에 후킹 함수에 무조건 WINAPI 이나 __stdcall 를 선언하여 호출 규약을 정해야 한다.

이제 hkDrawIndexedPrimitive 함수에 코드를 작성해 사물이 렌더링 되는 시점에 원하는 코드를 실행할 수 있다.



[사진 16] 후킹(Hooking) 전/후 비교

3.5 숨겨진 물체를 보이게 하기

렌더링할 때 숨겨진 물체가 보이게 하는 순서는 다음과 같다.

1. Z Buffer 비활성화

-DIP 함수를 통해 물체를 그리기 전에 Z Buffer를 비활성화해 물체가 벽 너머에서도 보일 수 있게 만든다.

2. oDrawIndexedPrimitive 호출

-Z Buffer가 비활성화된 상태에서 물체를 그리기 위해 원본 DrawIndexedPrimitive 함수를 호출한다.

3. Z Buffer 활성화

-Z Buffer를 다시 활성화해서 다른 물체가 정상적으로 그려질 수 있도록 한다.

Z Buffer를 활성화 또는 비활성화 시 SetRenderState 함수를 사용한다. 함수의 원형은 아래 그림과 같다.

```
HRESULT SetRenderState(
        D3DRENDERSTATETYPE State,
        DWORD              Value
);
```

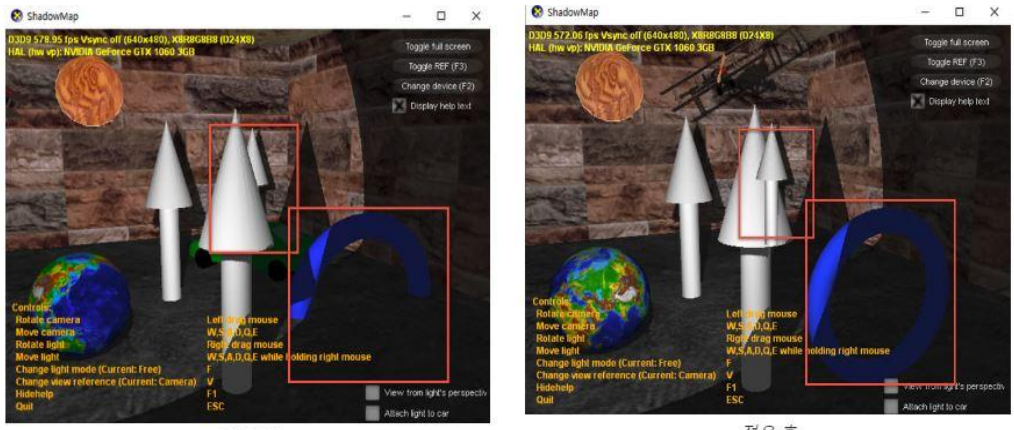
[사진 17] D3D9 기본 샘

이 함수의 첫 번째 인자에 D3DRS_ZENABLE를 주고 두 번째 인자에 D3DZB_TRUE 또는 D3DZB_FALSE 주는 것으로 Z Buffer를 활성화/비활성화 할 수 있다.

D3D9 기본 샘플에 적용해 보았다.

```
void __stdcall hkDrawIndexedPrimitive(LPDIRECT3DDEVICE9
pDevice, D3DPRIMITIVETYPE pType, INT BaseVertexIndex, UINT
MinVertexIndex, UINT NumVertices, UINT startIndex, UINT
primCount)
{
  pDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_FALSE); // Z
Buffer 비활성화
  // Drawing 전
  oDrawIndexedPrimitive(pDevice, pType, BaseVertexIndex,
MinVertexIndex, NumVertices, startIndex, primCount);
  // Drawing 후
  pDevice->SetRenderState(D3DRS_ZENABLE, D3DZB_TRUE); //Z
Buffer 활성화
}
```

[사진 18] D3D9 기본 샘플에 적용



[사진 19] D3D9 기본 샘플에 적용 전/후 비교

가려진 기둥과 바닥 속 물체가 보이는 걸 확인할 수 있다.

3.5 결론

핵 프로그램을 사용하는 유저들로 인해 유저들은 빠지게 되며 게임사에 많은 영향을 끼치고 있다. 이러한 현상들을 해결하기 위해 각 게임사마다 안티 치트를 도입하거나 개발하여 유저들의 불만을 덜어주고 게임의 질을 높이기 위해 노력하고 있으며, 유저들의 요구사항를 최대한 수용해 다양한 방법의 제재를 취하고 있다. 또한 게임 핵을 제작해 봄으로써 게임 핵을 생각보다 쉽게 만들 수 있고 이러한 문제점들은 지속적으로 이어질 것이기 때문에 더 업그레이드된 안티 치트 프로그램이나 지속적인 모니터링으로 조기에 빠르게 차단하는 것이 최선의 방법이다. 게임사에서는 유저들과의 주기적인 만남을 가질 수 있는 자리를 가져서 서로의 입장을 이해하고 더욱 발전될 수 있는 방향을 논의하는 것이 바람직하다고 판단된다.

참고문헌

[1] Detours: Binary Interception of Win32 Functions. Galen Hun, Doug Brubacher. Microsoft Research

[2] 온라인 게임에서의 게임 핵사용 실태에 관한 연구. 문현학. 청강문화산업대학교 게임콘텐츠. 한국컴퓨터정보학회 하계학술대회 논문집 제25권 제2호 (2017. 7)

[3] 온라인 게임 보안 동향, 안철수 연구소.

[4] 온라인 FPS 게임의 치팅 유형과 대응 방안에 관한 연구: 인도네시아 포인트 블랭크 사례를 중심으로. 임성진, 이대현. 한국산업기술대학교 산업경영대학원 디지털엔터테인먼트학과.

[5] 온라인 게임 내의 핵 프로그램 탐지 방법. 이창선. 상명대학교 경영학과. 2015년 춘계학술발표대회 논문집 제 22권 제 1호(2015. 4)

[6]그래픽 라이브러리 `DirectX`를 이용한 가려지는 물체 제거 기법(Occlusion Culling)에 관한 연구. 정성준. 서울대학교 대학원. 학위논문(석사)--서울대학교 대학원 :조선해양공학과2002.