HUNTING AN AUTO HUNTING PROGRAM

Homepage: https://sites.google.com/site/doc4code/

Email: sj6219@hotmail.com

2010/12/23

온라인 게임을 서비스 하다 보면, 자동 사냥 프로그램이 흥행에 영향을 끼치는 경우가 많다. 오토 프로그램이 너무 많으면, 게임의 흥미를 잃게 되어 결국 사용자가 감소된다.

이번 문서에서는 어떻게 게임 해킹을 막을 것인가에 대해서 고민해 보려고 한다. 막기만 하는 게 아니라, 적극적으로 공격하는 방법에 대해서도 알아보겠다.

온라인 게임을 해킹하는 게 쉬운 일은 아니다. 게임회사에서도 각종 보안 프로그램으로 막기 때문에, 상당한 해킹 기술이 있어야 한다. 게다가, 소스가 없는 상태에서 프로그램을 분석해야만 하기 때문에, 많은 시간과 노력이 필요하다.

해킹 프로그램 개발 회사도 돈이 있어야 운영되기 때문에, 인터넷에서 광고도 하고, 나름대로 매출을 늘리기 위해서 노력한다. 누구나 인터넷 검색을 하면 게임 해킹 프로그램을 쉽게 구할 수 있다. 게임 개발사도 마찬가지이다. 게임 개발사가 해킹 프로그램을 구해서 분석하면, 그 해킹 프로그램을 사용하는 유저를 쉽게 찾아낼 수 있다. 그 다음, IP 주소로 분류해보면, 어디가 작업장인지까지 알아낼 수 있다. 작업장은 해킹 회사의 주요 고객이기 때문에, 작업장을 단속하면 해킹 회사가 개발을 유지하기 힘들게 된다.

MEMORY MAPPED I/O

해킹 방지 알고리즘을 설명하기 전에, 그 기초가 되는 라이브러리 함수부터 구현해 보자.

우선 파일을 메모리로 읽어 들이는 클래스부터 만들어 보자. ReadFile() 함수를 이용해서 구현할 수도 있겠지만, Memory Mapped I/O 를 사용하는 편이 Hacking Program 에서 가로채기(Hooking)가 어려울 것이다.

```
class XFile
{
public:

        char    *m_pViewBegin;
        char    *m_pViewEnd;
        char    *m_pView;

        XFile( );
        XFile( void *pFile, DWORD size);
        void Open( void *pVoid, DWORD size);
        UINT Read( void *lpBuf, UINT uiCount);
        void Close( );
        ...
};

class XFileEx : public XFile
{
public:
        HANDLE m_hMapping;
        HANDLE m_hFile;

        XFileEx( );
        ~XFileEx( );
        BOOL    Open( LPCTSTR szPath);
        void    Close( );
};


XFile::XFile( )
{
        m_pViewBegin = 0;
        m_pView = 0;
        m_pViewEnd = 0;
}
```

```
XFile::XFile( void *pFile, DWORD size)
{
	m_pView = 0;
	m_pViewEnd = 0;

	Open( pFile, size);
}

void	XFile::Open( void *pVoid, DWORD size)
{
	m_pViewBegin = ( char *) pVoid;
	m_pView = m_pViewBegin;
	m_pViewEnd = m_pViewBegin + size;
}

UINT	XFile::Read( void *lpBuf, UINT uiCount)
{
	uiCount = min ( uiCount, ( UINT)( m_pViewEnd - m_pView));
	memcpy( lpBuf, m_pView, uiCount);
	m_pView += uiCount;
	return uiCount;
}

void	XFile::Close( )
{
	m_pView = 0;
	m_pViewEnd = 0;
}


XFileEx::XFileEx( )
{
	m_hFile = INVALID_HANDLE_VALUE;
	m_hMapping = 0;
}

XFileEx::~XFileEx( )
{
	Close( );
}
```

```
BOOL    XFileEx::Open( LPCTSTR szPath)
{
        UINT nSize;
        LPVOID  pMapView;
        m_hFile = ::CreateFile( szPath, GENERIC_READ, FILE_SHARE_READ, NULL,
                OPEN_EXISTING, 0, 0);
        if( m_hFile == INVALID_HANDLE_VALUE) {
                return FALSE;
        }
        m_hMapping = 0;
        nSize = GetFileSize( m_hFile, 0);

        m_hMapping = CreateFileMapping( m_hFile, NULL, PAGE_READONLY, 0, 0, 0);
        if ( m_hMapping == 0) {
                XFile::Open( 0, 0);
                return TRUE;
        }
        pMapView = MapViewOfFile( m_hMapping, FILE_MAP_READ, 0, 0, 0);
        if ( pMapView == 0)     {
                CloseHandle( m_hMapping);
                CloseHandle( m_hFile);
                m_hMapping = 0;
                m_hFile = INVALID_HANDLE_VALUE;
                return FALSE;
        }
        XFile::Open( pMapView, nSize);
        return TRUE;
}

void    XFileEx::Close( )
{
        if ( m_hMapping) {
                UnmapViewOfFile( m_pViewBegin);
                CloseHandle( m_hMapping);
                CloseHandle( m_hFile);
                m_hMapping = 0;
                m_hFile = INVALID_HANDLE_VALUE;
        }
        else if ( m_hFile != INVALID_HANDLE_VALUE) {
```

```
            CloseHandle( m_hFile);
            m_hFile = INVALID_HANDLE_VALUE;
        }
}
```

소스는 https://sites.google.com/site/doc4code/source/XLib.zip 의 XFile.cpp에 있다.

그 다음 만들어야 할 것은 Checksum 을 계산하는 함수이다.

```
DWORD UpdateCRC( DWORD seed, void *buf, int len);
```

여기서 seed 는 CheckSum 값을 구할 때 입력으로 사용된다.

seed 값이 조금이라도 다르면, 결과값도 많이 달라져서 결과값을 예측 불가능하게 구현 해야 한다.

소스는 https://sites.google.com/site/doc4code/source/XLib.zip 의 CRC.cpp에 있다.

실행 파일 변조

해커가 할 수 있는 가장 쉬운 방법은 실행(exe) 파일을 변조해서 새로운 실행 파일을 만드는 것이다. 실행 파일이 바뀌었는지 검사하는 것은 어렵지 않다.

아래 코드를 사용해서 파일이 바뀌었는지 검사해 볼 수 있다.

```
#define PROGRAM_SEED 0x12348921

DWORD CheckExe()
{
        DWORD dwCRC = 0;
        TCHAR szPath[MAX_PATH];
        GetModuleFileName( 0, szPath, sizeof( szPath));

        XFileEx file;
```

```
        if ( file.Open( szPath)) {

                dwCRC = UpdateCRC( PROGRAM_SEED, file.m_pView, file.GetLength( ));
        }
        return dwCRC;
}
```

위 프로그램을 클라이언트에서 실행한다. 그 다음 dwCRC 를 서버로 보낸다. 서버에서도 똑 같은 방식으로 계산해서 두 결과 값을 비교해 보면, 실행 파일이 변조되었는지 알 수 있다.

Dll 파일의 변조도 마찬가지로 검사해 보면 된다.

해킹 프로그램 탐지

디스크에 저장된 실행 파일을 변조하면 쉽게 탐지되기 때문에, 대부분의 해커는 실행 파일을 변조하지 않는다. 그대신, dll 을 게임 클라이언트에 삽입시켜서, 클라이언트가 실행되면서 그 dll 도 함께 실행되는 형태로 되어 있다. 경우에 따라서는 메모리에 로딩된 코드를 변조하기도 한다.

해킹 업체마다 자신들만의 방식이 있어서, 어떤 곳은 데이터를 읽거나 OS 함수를 가로채기만 하는 곳도 있고, 어떤 곳은 클라이언트 코드까지 적극적으로 변경하는 곳도 있다. 또, 어떤 곳은 게임 화면을 패턴 인식함으로써 자동 사냥을 구현하기도 한다.

그런데, 단순히 패턴 인식만으로는 해킹하면 제한이 많아서, 제공하는 기능이 떨어지고, 성능도 나빠져서 게임하기가 불편해진다. 그래서, 대부분의 해킹 프로그램은 exe 파일이나 dll 파일의 형식으로 존재한다. 클라이언트에서 실행 중인 exe 파일을 검사하거나, 클라이언트에 로딩된 dll 파일을 검사하면 해킹 프로그램을 탐지할 수 있다.

해킹 프로그램을 탐지 할 수 있는 가장 쉬운 방법은 파일 이름으로 찾는 것이다. 또는 파일의 checksum 을 검사해 볼 수도 있다.

그렇지만, 버전 업이 되거나 해킹 업체에서 파일 이름을 바꾸면 소용이 없어진다.

그래서, 내가 시도한 방법은 그 프로그램의 고유한 특징을 검사하는 것이다.

나는 exe 파일(또는 dll 파일)에 들어있는 아이콘(icon)을 검사했다. 해커는 내가 icon 을 검사하는지 모르기 때문에, 다음 버전 업데이트를 하더라도 icon 을 바꾸지 않는다. 그러므로, 해킹 프로그램이 버전 업 되더라도 계속 오토 프로그램을 탐지할 수 있다.

다음은 dll 파일의 첫 번째 아이콘의 checksum 을 구하는 코드이다.

CheckDll() 함수를 이용해 checksum 을 구한다.

```
#define GetImgDirEntryRVA( pNTHdr, IDE ) ₩
        (pNTHdr->OptionalHeader.DataDirectory[IDE].VirtualAddress)
#define GetImgDirEntrySize( pNTHdr, IDE ) ₩
        (pNTHdr->OptionalHeader.DataDirectory[IDE].Size)


DWORD CheckDll(HMODULE hModule)
{
        TCHAR ModName[MAX_PATH];
        GetModuleFileName((HINSTANCE) hModule, ModName, MAX_PATH);
        XFileEx file;
        file.Open(szPath);

        PIMAGE_DOS_HEADER DosHeader = (PIMAGE_DOS_HEADER) file.m_pView;
        if (DosHeader->e_magic != IMAGE_DOS_SIGNATURE)
                return 0;
        IMAGE_NT_HEADERS *NTHeader = (IMAGE_NT_HEADERS *)
                ((char *)DosHeader          + DosHeader->e_lfanew);
        if (IMAGE_NT_SIGNATURE != NTHeader->Signature) {
                return 0;
        if ((NTHeader->FileHeader.Characteristics & (IMAGE_FILE_32BIT_MACHINE |
        IMAGE_FILE_DLL)) != IMAGE_FILE_32BIT_MACHINE)
                return 0;

        return DumpResourceSection((DWORD_PTR) file.m_pView, NTHeader);
}

DWORD DumpResourceSection(DWORD_PTR base, PIMAGE_NT_HEADERS pNTHeader)
{
        DWORD nResEntries = 0;
```

```
        PIMAGE_RESOURCE_DIRECTORY_ENTRY pResEntriess = 0;

        DWORD resourcesRVA;
        PIMAGE_RESOURCE_DIRECTORY resDir;

        resourcesRVA = GetImgDirEntryRVA(pNTHeader,
                IMAGE_DIRECTORY_ENTRY_RESOURCE);
        if ( !resourcesRVA )
                return 0;

        resDir = (PIMAGE_RESOURCE_DIRECTORY)
                GetPtrFromRVA( resourcesRVA, pNTHeader, base );

        if ( !resDir )
                return 0;

        DumpResourceDirectory(resDir, (DWORD_PTR)resDir, 0, 0, nResEntries, pResEntriess);
        return DumpResTable(base, pNTHeader, (DWORD_PTR)resDir, nResEntries, pResEntriess);
}

void DumpResourceDirectory
(
        PIMAGE_RESOURCE_DIRECTORY resDir,
        DWORD_PTR resourceBase,
        DWORD level,
        DWORD resourceType,
        DWORD &nResEntries,
        PIMAGE_RESOURCE_DIRECTORY_ENTRY &pResEntriess
)
{
        PIMAGE_RESOURCE_DIRECTORY_ENTRY resDirEntry;
        UINT i;

        resDirEntry = (PIMAGE_RESOURCE_DIRECTORY_ENTRY)(resDir+1);

        if ( level == 1 && (resourceType == (WORD) RT_ICON))
        {

                pResEntries = resDirEntry;
                nResEntries = resDir->NumberOfIdEntries;
                return;
```

```
        }


        for ( i=0; i < resDir->NumberOfNamedEntries; i++, resDirEntry++ )
                DumpResourceEntry(resDirEntry, resourceBase, level+1, nResEntries,
pResEntriess);

        for ( i=0; i < resDir->NumberOfIdEntries; i++, resDirEntry++ )
                DumpResourceEntry(resDirEntry, resourceBase, level+1, nResEntries,
pResEntriess);
}

void DumpResourceEntry
(
        PIMAGE_RESOURCE_DIRECTORY_ENTRY resDirEntry,
        DWORD_PTR resourceBase,
        DWORD level
        DWORD &nResEntries,
        PIMAGE_RESOURCE_DIRECTORY_ENTRY &pResEntriess
)
{

        if ( resDirEntry->OffsetToData & IMAGE_RESOURCE_DATA_IS_DIRECTORY )
        {
                DumpResourceDirectory((PIMAGE_RESOURCE_DIRECTORY)
                        ((resDirEntry->OffsetToData & 0x7FFFFFFF) + resourceBase),
                        resourceBase, level, resDirEntry->Name, nResEntries, pResEntriess);
                return;
        }
}

DWORD DumpResTable(
        DWORD_PTR base,
        PIMAGE_NT_HEADERS pNTHeader,
        DWORD_PTR resourceBase,
        PIMAGE_RESOURCE_DIRECTORY_ENTRY pStrResEntry,
        DWORD cStrResEntries,
        DWORD nResEntries,
        PIMAGE_RESOURCE_DIRECTORY_ENTRY pResEntriess
)
{
```

```
        DWORD dwHint = 0;

        for (unsigned i = 0; i < cStrResEntries; i++, pStrResEntry++) {
                PIMAGE_RESOURCE_DATA_ENTRY pResDataEntry
                        = GetDataEntryFromResEntry(resourceBase, pStrResEntry );

                char* pResEntry = (char *)GetPtrFromRVA(pResDataEntry->OffsetToData,
                        pNTHeader, base );
                if (pResEntry)
                        dwHint = UpdateCRC(0, pResEntry, pResDataEntry->Size);
                break;
        }
        return dwHint;
}

PIMAGE_RESOURCE_DATA_ENTRY GetDataEntryFromResEntry(
        DWORD_PTR resourceBase,
        PIMAGE_RESOURCE_DIRECTORY_ENTRY pResEntry )
{
        PIMAGE_RESOURCE_DIRECTORY pStupidResDir;
        pStupidResDir = (PIMAGE_RESOURCE_DIRECTORY)
                                        (resourceBase + pResEntry->OffsetToDirectory);

        PIMAGE_RESOURCE_DIRECTORY_ENTRY pResDirEntry =
                (PIMAGE_RESOURCE_DIRECTORY_ENTRY)(pStupidResDir + 1);// PTR MATH

        PIMAGE_RESOURCE_DATA_ENTRY pResDataEntry =
                (PIMAGE_RESOURCE_DATA_ENTRY)(resourceBase +
                                        pResDirEntry->OffsetToData);

        return pResDataEntry
}


LPVOID GetPtrFromRVA(
        DWORD rva,
        PIMAGE_NT_HEADERS pNTHeader,
        DWORD_PTR imageBase )
{
        PIMAGE_SECTION_HEADER pSectionHdr;
        INT delta;
```

```
        pSectionHdr = GetEnclosingSectionHeader( rva, pNTHeader );
        if ( !pSectionHdr )
                return 0;

        delta = (INT)(pSectionHdr->VirtualAddress-pSectionHdr->PointerToRawData);
        return (PVOID) ( imageBase + rva - delta );
}

PIMAGE_SECTION_HEADER GetEnclosingSectionHeader(DWORD rva,

        PIMAGE_NT_HEADERS pNTHeader)
{
        PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(pNTHeader);
        unsigned i;

        for ( i=0; i < pNTHeader->FileHeader.NumberOfSections; i++, section++ )
        {
                if ( (rva >= section->VirtualAddress) &&
                                (rva < (section->VirtualAddress + section->Misc.VirtualSize)))
                        return section;
        }

        return 0;
}
```

자세한 것은 PE(Portable Executable) Format 을 공부해야 한다. 복잡한 것 같지만, 단순히 파일 구조만 알면 된다. 파일에서 icon 을 저장하는 위치를 찾아서, checksum 을 구한다.

코드 변조

해킹 프로그램을 탐지함으로써, 마음대로 오토 사용자를 구별해 낼 수 있게 되었다.

해킹 업체는 서비스를 중단했고, 싸움은 나의 일방적인 승리로 끝나는 듯 했다.

그렇지만, 해커도 쉽게 물러서지 않았다. 이제 클라이언트 코드를 변조하기 시작했다. 메모리에 로딩되어 있는 코드를 변조하여 해킹 탐지 코드를 건너뛰게 만들었다.

놀랍게도 해커가 그 위치를 정확하게 찾아낸 것이다. 파일 자체는 건드리지 않고, 프로그램이 로딩된 후, 메모리 상의 코드만 변조했다. 그래서, 앞에서 설명한 방법들은 무용지물이 되었다.

하지만, 나에게 해커와의 싸움은 정말 즐거운 게임이었다. 어떤 게임도 이렇게 재미있을 순 없었다. ㅎㅎ

이번엔, 메모리 상의 코드와 디스크 상의 코드를 비교해서 검사하는 코드를 추가했다.

```
BOOL    CheckExe()
{

        TCHAR szPath[MAX_PATH];
        GetModuleFileName(0, szPath, sizeof(szPath));

        XFileEx file;
        if   (!file.Open(szPath))
                return FALSE;

        PIMAGE_DOS_HEADER DosHeader = (PIMAGE_DOS_HEADER) file.m_pView;
        IMAGE_NT_HEADERS *pNTHeader = (IMAGE_NT_HEADERS *)
                ((char *)DosHeader        + DosHeader->e_lfanew);
        PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(pNTHeader);
        unsigned i;

        for ( i=0; i < pNTHeader->FileHeader.NumberOfSections; i++, section++ )
        {
                if (strcmp((char *) section->Name, ".text") != 0)
                        continue;
                char *pFile = file.m_pView + section->PointerToRawData;
                char *pMemory = (char *) GetModuleHandle(0) + section->VirtualAddress;
```

```
                DWORD dwSize = section->Misc.VirtualSize;
                if (memcmp(pFile, pMemory, dwSize) == 0)
                        return TRUE;
                return FALSE;

        }
        return FALSE;
}
```

또, 코드를 바꾸지 못하도록 한가지 더 추가했다. 해킹 탐지 코드를 dll 파일로 만들었다. 보통 때는 메모리 상에 없다가 실행 되기 전에 LoadLibrary() 되고, 실행 후에 FreeLibrary() 되도록 변경했다. 그래서, 해커가 코드를 변조하기 힘들게 되었다. 메모리 상에 코드가 있어야 고칠 수 있는데, 아예 코드가 없어서 변조할 수 조차 없다. 변조하려면 LoadLibray() 시점에서 가로챈 후, 그 후에 코드를 변조해야 한다.

서버 연계

또 한동안 조용해졌다. 한국의 해킹 업체에서 우리 게임은 포기해 버렸다. 그런데, 이번엔, 중국의 게임업체가 나타났다. 중국 업체는 한국 업체보다 실력이 좋았다. 아무리, 해킹 업체라고 하지만, 중국보다 못하다니. 기분이 좋지 않았다.

중국업체는 클라이언트에 있는 해킹 방지 코드 들의 위치를 정확하게 찾아냈다. 놀라운 일이었다. 하지만, 해킹 탐지 자체를 막지는 못했다. 그대신, 클라이언트에서 서버로 패킷(packet)을 보내는 순간을 변조했다. 그 순간, 해킹 정보 패킷인지 검사해서 해킹 정보를 보내지 못하도록 했다.

나는 더 근본적으로 막을 수 있는 방법을 고민했다. 그래서, 서버의 도움을 받기로 했다.
클라이언트만으로는 해커와의 싸움에서 이기기 힘들었다.

서버에서 클라이언트 파일을 등록하도록 변경했다. 그리고, 코드 변조 방지 코드를 서버에도 만들어서, 클라이언트와 주기적으로 비교하게 하였다.

이것만으로는 부족한 것 같아, 지금까지 만들었던 여러 해킹 방지 코드들을 여러 부분으로 흩어 놓았다. 일부는 서버와 연계되도록 되어 있어서 해킹 방지 코드인지 알기 힘들게 해놓았다. 그리고, 이런 해킹 방지 코드들이 서로 서로 보호하도록 했다. 마치 체스에서 룩이 비숍을 보호하고, 비숍이 나이트를 보호하고, 나이트가 룩을 보호하는 것과 같다. 그래서, 모든 해킹 수수께끼를 한꺼번에 풀어야만 되도록 했다. 한 쪽만 풀면 다른 쪽에서 변조된 것을 찾도록 했다.

결론

중국업체에서도 포기해 버렸다. 그사이 우리 게임의 동접자가 많이 떨어졌다. 중국업체에서 해킹하기 힘들고, 매출도 작은 게임을 유지하기 힘들다고 판단한 것 같다. 중국업체랑 싸우면서, Anti-Hacking 기술을 개발할 수 있었는데..... 재밋거리가 하나 줄었다. 아쉽다. 쩝.