AhnLab HackShield

Quick Guide

Version 2.0

AhnLab HackShield for Online Game

基本功能

AhnLab

1 前言

- 本文档是使用HackShield Pro(以下简称为Hackshield) SDK的指导文档.以Microsoft Visual C++ 6.0环境中C/C++为基准.
- 为帮助第一次使用HackShield，操作大概的说明和简单的代码.
- 在\Sample文件夹里提供MiniA代码. 请参考代码.
- 在\Doc文件夹里提供 *"AhnLab_HackShield_2.0_Programming_Guide"*文档.此文档为Hackshield Pro的整体结构和功能以及所有的API函数的使用说明.详细的内容请参考此文档.

2 HackShield Pro SDK结构

- \Bin：Hackshield运行时相关文件.
- \Data : HackShield 使用的Date文件。
- \Developer： 开发人员测试HackShield所需要的文件。
- \Doc： 使用说明书.
- \Include：Hackshield头文件.
- \Lib：Hackshield库文件.
- \PatchSet ： Hackshield更新时的更新包文件.
- \Sample :样本程序.

3 使用前准备事项

- 导入库文件.
  - ✓ Hackshield使用Windows的基本库文件version.lib, winmm.lib.请在项目中添加这2个库文件.
  - ✓ \Lib\HShield.lib文件为必要库文件.复制到相关位置后在项目中导入.
- \Include\HShield.h文件为必要的头文件.复制到相关为之后在项目中导入.
- Hackshield序列号.由4位游戏代码以及24位文字组成序列号.此序列号为每个游戏固定的序列号,并不通用.

4 使用Hackshield

- 在游戏项目中调用HShield.h文件.

```
#include "HShield.h"
```

- 游戏的Enter-Point函数上增加调用HackShield服务的函数.
  - ✓ HackShield代码的使用是以初始化(Initialize) → 开始(Start) → 执行游戏 → 停止(Stop) → 释放(Uninitialize)的顺序进行. 游戏客户端初始化之前执行HackShield Module的初始化和开始， 结束时必须要执行停止和释放。

```
WinMain (...)
{
    if ( !HS_Init() )
```

```
    {
        HS_UnInit();                                    HackShield Initialize 部分
        return 0;
    }

    if ( !HS_StartService() )
    {
        HS_StopService();
        HS_UnInit();                                    HackShield Service Start 部分
        return 0;
    }

    // 游戏运行程序部分
    …

    HS_StopService();                                   HackShield Service Stop 部分

    HS_UnInit();                                        HackShield Uninitialize 部分
}
```

✓ Uninitialize未被正常调用的情况下重新执行代码时有可能会不能正常执行. Initialize(Start)失败时必须处理为调用 Uninitialize(Stop).

✓ 为游戏客户端的安全保护HackShield初始化结束的话最好马上开始HackShield服务.

❖ Hackshield初始化(_AhnHS_Initialize)函数.

```
BOOL HS_Init()
{
    int         nRet = 0;
    TCHAR       szFullFilePath[MAX_PATH];
    TCHAR       szMsg[MAX_PATH];
    DWORD       dwOption = 0;

    // ①指定HackShield文档下的EhSvc.dll文件的位置.
    lstrcat ( szFullFilePath, _T( "\\HShield\\EhSvc.dll" ) );

    // ②设定_AhnHS_Initialize函数中调用的选项
    dwOption = AHNHS_CHKOPT_ALL ;

    // ③ 调用_AhnHS_Initialize函数开始Hackshield服务.
    nRet = _AhnHS_Initialize ( szFullFilePath,
                               HS_CallbackProc,          // CallBack函数
                               1000,                     // 游戏代码
                               "B228F291B7D7FAD361D7A4B7",   //序列号
                               dwOption,                 //Hacksheld选项
                               AHNHS_SPEEDHACK_SENSING_RATIO_NORMAL );

    // ④ 检查_AhnHS_Initialize函数的返回值并处理报错信息.
```

```
    if ( nRet != HS_ERR_OK )
    {
        switch( nRet )
        {
        case HS_ERR_COMPATIBILITY_MODE_RUNNING:
        case HS_ERR_NEED_ADMIN_RIGHTS:
        case HS_ERR_INVALID_FILES:
        case HS_ERR_INIT_DRV_FAILED:
        case HS_ERR_DEBUGGER_DETECT:
        case HS_ERR_NOT_INITIALIZED:
        default:
            wsprintf( szMsg, "HS_Init() Error.(%x)", nRet );
            break;
        }
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        return FALSE;
    }
    return TRUE;
}
```

① szFullFilePath 变量上设置HackShield SDK里 EhSvc.dll 文件的绝对路径. 该文件夹一般位与执行文件路径上将要生成的HShield 文件夹下面. 为了正确地设置绝对路径，最好利用GetModuleFileName函数. GetCurrentDirectory函数有可能不能获得所需要的路径.

② 在这里定义的Flag在调用_AhnHS_Initialize函数时被使用为第5个参数. 设置HackShield的基本功能. 为了游戏的编译，要删除几个选项之后执行. 对各Flag的详细事项请参考 *"AhnLab HackShield 2.0 _Programming_Guide: 2.4. Application Programming Interface"*. 如果没有特别的情况，测试环境及分发时推荐直接使用上面使用的Flag

③ 调用_AhnHS_Initialize 函数的时候需要游戏的代码和HackShield序列号. 最后的参数是对加速外挂感知的敏感度设置. 加速外挂的感知程度根据游戏公司的政策决定. 一般情况下会使用 AHNHS_SPEEDHACK_SENSING_RATIO_NORMAL。对各参数的仔细事项请参考 "程序向导: 2.3. Application Programming Interface".

④ 调用_AhHS_Initialize 函数的返回值，如果不是正常的执行(HS_ERR_OK)的话，将根据相关的错误代码提示相应的报错信息并进行终止. 对各种情况的错误提示请参考"AhnLab HackShield Programming Guid"之后对各报错的特性与各游戏公司的政策进行操作.

ex) HS_ERR_INVALID_FILES的情况:

```
wsprintf( szMsg, "安装错误的文件.\n请重新安装程序. (%x)", nRet );
break;
```

❖ 开始Hackshield服务的函数 (_AhnHS_StartService).

```
BOOL HS_StartService()
{
    int         nRet = 0;
    TCHAR       szMsg[MAX_PATH];

    // ① 调用_AhnHS_StartService函数开始服务.
```

```
    nRet = _AhnHS_StartService();

    // ② 检查_AhnHS_StartService函数的返回值并对报错进行处理.
    if ( nRet != HS_ERR_OK )
    {
        switch ( nRet )
        {
        case HS_ERR_START_ENGINE_FAILED:
        case HS_ERR_DRV_FILE_CREATE_FAILED:
        case HS_ERR_REG_DRV_FILE_FAILED:
        case HS_ERR_START_DRV_FAILED:
        default:
            wsprintf ( szMsg, " HS_StartService()_Error (%x)", nRet );
            break;
        }
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        return FALSE;
    }
    return TRUE;
}
```

① 为了使用HackShield的服务，之前必要要调用Initialize 函数. StartService 函数被调用之后才能对Hacking进行防御，所以最好在执行游戏初始化工作之前调用_AhnHS_StartService函数进行HackShield的服务.

② 根据_AhHS_StartService函数的返回值确认有没有正常被调用，如果未成功(HS_ERR_OK)，将会根据相应的错误代码进行提示错误消息并终止程序. 对各情况的错误提示参考 “AhnLab HackShield Programming Guid”之后进行操作.

❖ 停止Hackshield服务的函数. (_AhnHS_StopService)

```
BOOL  HS_StopService()
{
    Int  nRet = 0;

    // ① 调用_AhnHS_StopService函数，停止HackShield服务.
    nRet = _AhnHS_StopService();

    if ( nRet != HS_ERR_OK )
    {
        return FALSE;
    }
    return TRUE;
}
```

① 停止Hackshield服务必须是在Hackshield工作过程中调用停止函数.调用_AhnHS_StopService函数会停止HackShield服务，将会停止对外挂的阻止功能和外挂探知功能.

❖ 终止Hackshield服务的函数. (_AhnHS_Uninitialize)

```
BOOL HS_UnInit()
{
    int  nRet = 0;

    // ① 调用_AhnHS_Uninitialize函数结束Hackshield服务.
    nRet = _AhnHS_Uninitialize();

    if ( nRet != HS_ERR_OK )
    {
        return FALSE;
    }
    return TRUE;
}
```

① 游戏客户端程序非正常结束的话有可能不能重新加载HackShield驱动.这种情况下重新执行时会发生不能正常执行的情况，所以结束服务时必须执行 Uninitialize. 客户端的正常结束是必然的，调用callback函数之后和非正常结束的 exception处理上也要必须执行 Stop和 Uninitialize.

② 对于非正常结束的情况，在游戏最开始的位置添加下列代码，就会执行因异常而引起的游戏结束时，HackShield的终止例行程序。

```
::SetUnhandledExceptionFilter ( Game_UnhandledExceptionHandler );
```

③ 在上面的 Game_UnhandledExceptionHandle()函数内处理为调用HackShield StopService与Uninitialize即可。当然，该情况是针对于发生异常时的情况，不发生异常而正常结束的情况时是合适的。

❖ 与拦截外挂功能及外挂感知功能相关的事件传达函数. (_AhnHS_Callback)

```
int __stdcall HS_CallbackProc ( long lCode, long lParamSize, void* pParam )
{
    TCHAR       szMsg[MAX_PATH];

    // ①对各种情况，输出正确的错误提示
    switch ( lCode )
    {
    // ② Engine Callback
    case AHNHS_ENGINE_DETECT_GAME_HACK:
        wsprintf( szMsg, "以下程序不能和游戏一起运行. (%x) \n [%s]",
                lCode, (LPTSTR)pParam );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ③ AutoMacro感知
    case AHNHS_ACTAPC_DETECT_AUTOMACRO:
        wsprintf(szMsg, _T("感知到有宏功能的程序在运行. (Code = %x"),
                lCode);
        MessageBox(NULL, szMsg, szTitle, MB_OK);
        break;
    // ④ 不做另外处理
```

```
    case AHNHS_ACTAPC_DETECT_AUTOMOUSE:
    case AHNHS_ACTAPC_DETECT_ALREADYHOOKED:
        break;
    // ⑤ Speed相关
    case AHNHS_ACTAPC_DETECT_SPEEDHACK:
    case AHNHS_ACTAPC_DETECT_SPEEDHACK_APP:
        wsprintf( szMsg, "感知到有加速程序在运行. (%x)", lCode );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ⑥ 防止Debugging
    case AHNHS_ACTAPC_DETECT_KDTRACE:
    case AHNHS_ACTAPC_DETECT_KDTRACE_CHANGED:
        wsprintf( szMsg, "感知到有反编译的企图. (%x)", lCode );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ⑦ 其他的外挂防止功能
    case AHNHS_ACTAPC_DETECT_DRIVERFAILED:
    case AHNHS_ACTAPC_DETECT_HOOKFUNCTION:
    case AHNHS_ACTAPC_DETECT_MODULE_CHANGE:
    case AHNHS_ACTAPC_DETECT_LMP_FAILED:
    case AHNHS_ACTAPC_DETECT_MEM_MODIFY_FROM_LMP:
    case AHNHS AHNHS_ACTAPC_DETECT_ENGINEFAILED:
    case AHNHS_ACTAPC_DETECT_CODEMISMATCH:
    case AHNHS_ACTAPC_DETECT_ANTIFREESERVER:
    case AHNHS_ACTAPC_DETECT_ABNORMAL_HACKSHIELD_STATUS:

        wsprintf( szMsg, "探测到外挂程序和拦截程序，请关闭后运行. (%x)", lCode );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    }
    return 1;
}
```

① HackShield负责对感知信息的处理。以符合游戏rule的标准对各情况进行定义. 对各情况的详细内容请参考"AhnLab HackShield Programming Guid: 2.3. Application Programming Interface".

② 外挂程序或有可能对游戏运行造成影响的程序的感知情况. 通过pParam可以知道感知的程序名并把该程序名告诉使用者. 显示提示后请不要让游戏客户端继续进行，而是终止它。

③ Hackshield对AutoMacro的感知情况.请结束游戏客户端的运行.

④ 此情况下最好不要进行另外处理.根据需要只生成相关的日志文件即可.出现AHNHS_ACTAPC_DETECT_ALREADYHOOKED时是部分API函数已经被Hooking的情况，有时与安全产品软件一起也会对正常的程序进行Hooking. 出现AHNHS_ACTAPC_DETECT_AUTOMOUSE的时候有时不是Detection，而是Protection功能，所以 Callback函数不能被调用.

⑤ 与使用加速外挂类似的情况即系统的时间变化速度不正常的情况.Speedhack的可能性很高，这时提示错误消息并根据游戏开发公司的策略判断结束与否.

⑥ 发生DebugTrace的情况. 很有可能是对游戏的程序进行调试，所以请提示报错消息之后，结束游戏

客户端的进行。

⑦ Message Hooking或修改模块等类似的功能上发生异常的情况. 请提示报错消息只后结束游戏客户端的进行。

5 使用时注意问题

- 上述中提到过的Initialize时，使用的HackShield选项是可以根据游戏开发公司的政策进行选择使用的。 为了更安全地保护游戏客户端而鼓励使用HackShield的全部属性选项.
- 为了Debugging与客户支持，最好显示报错信息时把ErrorCode也一起显示.
- 处理报错Message时，举例使用了MessageBox(NULL, …)，但在全屏的游戏里报错消息很有可能会出现在游戏画面的后面，可能的话望通过游戏的UI进行处理.使用MessageBox 函数的话,代替NULL处理Handle.
- 出现问题时如果只显示消息框的情况下游戏可以继续运行,所以在出现问题的情况下除了显示消息框外请强制停止游戏的运行.

6 测试和分发

- Hackshield使用后重新编译游戏会重新生成一个游戏客户端.
- 在游戏的文件夹下面生成一个HShield文件夹.
  - ✓ ex) …\[Game Directory]\HShield
- 将得到的 \Bin\HShield 文件夹下的所有文件复制到之前生成的 HShield 文件夹里.
- 执行游戏后可以通过以下几种简单的方式确认Hackshield是否正常运行.
  - ✓ Case 1. 执行Process Explorer，查看相关dll信息是否出现。
  - ✓ Case 2. 重新命名HShield里的EhSvc.dll文件确认能够被感知
  - ✓ Case 3. 使用外挂并确认是否能正常的感知外挂
- 如果没有异常则对游戏执行程序进行加壳.
  - ✓ 使用(Packer)加壳工具可以防止轻易地对游戏执行程序进行debugging.
  - ✓ HackShield提供upx加壳.
  - ✓ ex) upx.exe –f Source.exe –o Output.exe
- 如果没有异常可以进行服务器连动的部分 (参考服务器连动Quick Guide).
- 最初分发时请如下进行.
  - ✓ 使用及测试 ⇒ Quality Assurance (Ahnlab)进行 ⇒ 传达QA Report后确认是否有修改事项然后确认最终分发的版本⇒分发
  - ✓ 分发的时候最好删除在测试过程中生成的hshield.log文件.