AhnLab HackShield

Quick Guide

Version 2.0

AhnLab HackShield for Online Game

扩展服务器联动功能

AhnLab

1 开始前

- 本文件是为了使用HackShield Pro(以外称为HackShield) SDK的GuidLine.
  在Microsoft Visual C++ 6.0环境下以C/C++语言制作的。下面对扩充服务器联动功能的部分进行介绍。

- 为了让大家理解如何使用HackShield，进行大概的说明以及简单的Sample代码演示。
  细节体现的部分请按照各游戏开发公司的政策进行。

- \ Doc文档里提供”AhnLab HackShield Pro Programing Guid” 文件。此文件说明并包含HackShield Pro 的全部构造与API函数使用方法。请参考。

2 HackShield Pro 扩充服务器联动功能的结构文件

- \Bin\AntiCrack\AntiCpXSvr.dll：扩充服务器联动 dll 文件.
- \Bin\AntiCrack\HSBGen.exe： 生成.hsb 文件的工具
- \Bin\AntiCrack\HSPub.key：服务器联动认证key文件
- \Bin\HShield\3n.mhe：Heuristic引擎版本管理文件
- \Bin\HShield\hshield.dat :：HackShield版本管理文件
- \Include\AntiCpXSvr.h：扩充服务器联动头文件
- \Lib\AntiCpXSvr.lib： 扩充服务器联动 Library File
- \Bin\AntiCrack\Linux\libanticpxsvr.so：扩充服务器联动，动态Library File (Linux)
- \Bin\AntiCrack\Solaris\libanticpxsvr.so：扩充服务器联动，动态Library File (Solaris)
- \Lib\Linux\libanticpxsvr_st.a：扩充服务器联动，静态Library File (Linux)
- \Lib\Solaris\libanticpxsvr_st.a：扩充服务器联动，静态Library File (Solaris)

※ Linux 与 Solaris 文件只支持 32bit x86 平台.

3 运行结构

- HackShield 扩充服务器联动的通信结构将按照现有游戏中体现的通信结构进行使用。（不需要另外搭建通信模块）

- 对客户端的版本进行确认来防止不正当的行为。

  - ✓ 如果游戏客户端不是被允许的版本，则断开连接直到安装最新的补丁

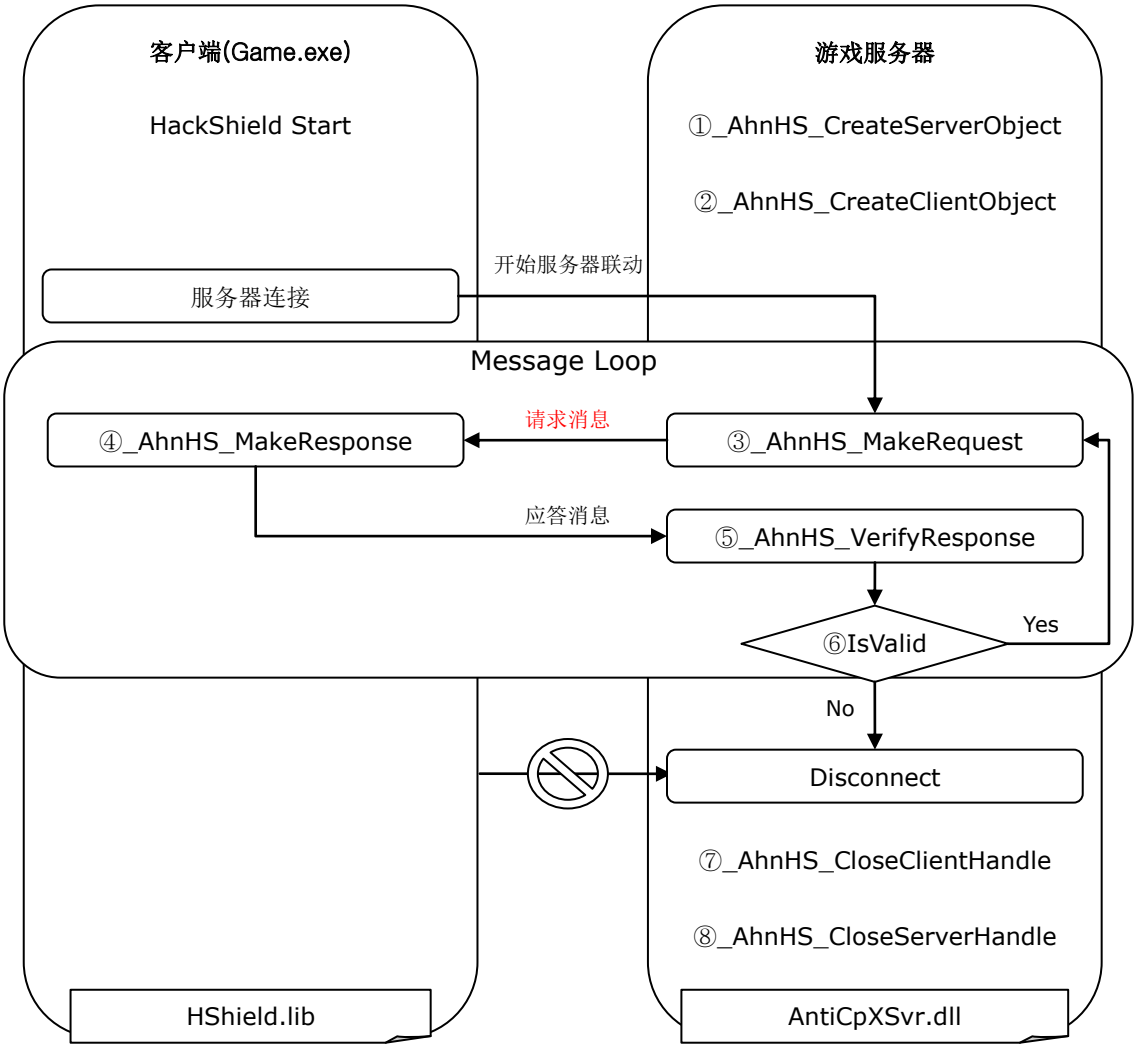
- 对客户端的执行文件，内存以及HackShield模块进行判断是否被侵犯。

  - ✓ 按照规则，以一段时间为间隔确认发送的请求消息与对此的回应消息。

  - ✓ 对发送请求的消息，回答的不正确则判断为被黑客侵袭，从而断开连接

✓ 如对请求消息没有做出回答，也将被判断为被黑客攻击，从而断开连接。

❖ 全部的运行结构如下



① 游戏服务器起初运行时 ， 通过_AhnHS_CreateServerObject函数生成AHNHS_SERVER_HANDLE。 生成的这个Handle应维持到游戏服务器终止为止。（参考：下一阶段制作Client Handle时需要使用Server Handle。）

② 每个客户端连接时，将_AhnHS_CreateClientObject函数与Server Handle作为参数来生成AHNHS_CLIENT_HANDLE。这个Handle将维持到客户端与网络断开为止。（参考：Client Handle将被使用在下一阶段制作请求消息。）

③ 为了监视客户端是否被篡改，服务器端将生成请求消息并发送到客户端。请求消息将使用_AhnHS_MakeRequest函数来生成，并将Client Handle作为参数。

④ 客户端会应服务器的请求消息做出相应的应答消息。应答消息通过_AhnHS_MakeResponse函数生成。

⑤ 检查客户端的应答消息是否有效。应答消息的有效性检查是通过_AhnHS_VerifyResponse 函数进行的。

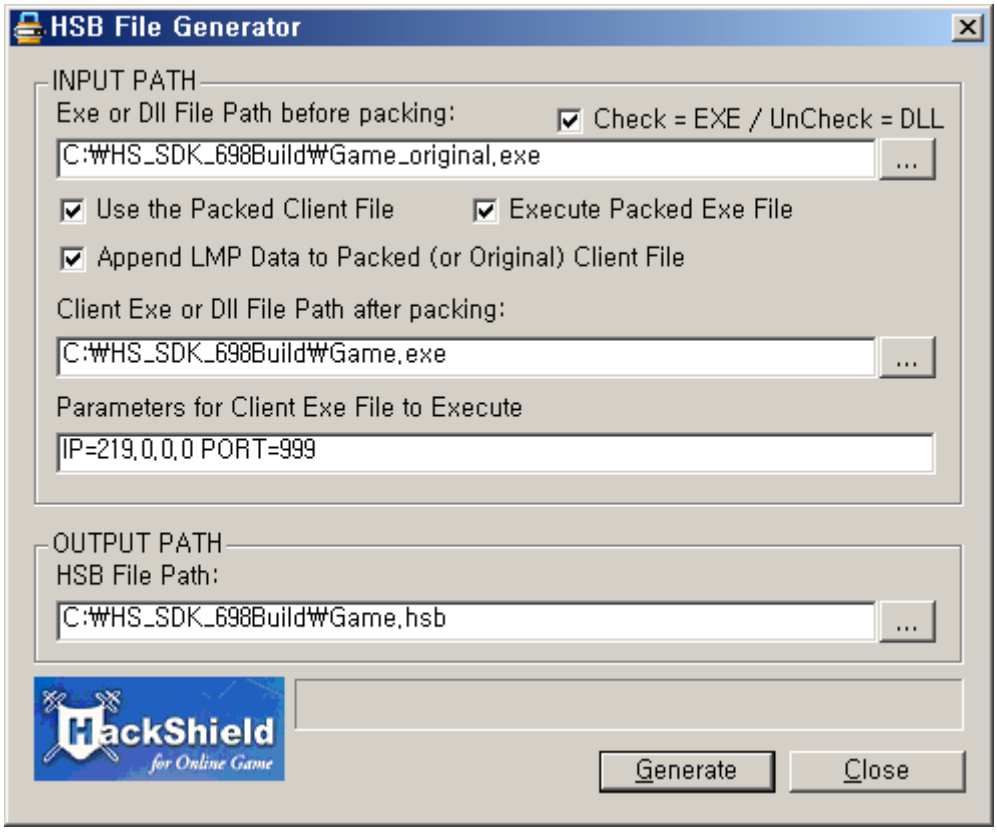
⑥ 在_AhnHS_VerifyResponse函数中发生除ERROR_SUCCESS值以外的错误代码时，根据错误代码做出适当的处理，然后与游戏客户端中断连接。

⑦ 与客户端断开连接时（session终止时），关掉在第二阶段生成的Client Handle.

⑧ Server Process终止时关掉第一阶段生成的Server Handle.(此时Client Handle必须全部被关掉)

4 运用前的准备事项

- 利用\Bin\AntiCrack\HSBGen.exe来生成 .hsb文件.
  - ✓ 注意事项!!! 客户端执行文件必须是加壳之前的原先的release文件。
  - ✓ 使用HSBGen.exe生成AntiCpX.hsb 文件的方法如下.



1. 对象文件是EXE文件时选择Check,对象文件是DLL是UnCheck.

2. 在Exe or Dll File Path before packing 输入框里设置加壳之前的原执行文件的路径。 通过点击［…］按钮来选择文件。

3. 想要分发加壳的客户端执行文件，则在之前说明的第一个输入框里输入加壳之前的文件路径。

4. 选择Use the Packed Client File项，然后在Client Executable File Path after packing输入框里输入加壳的文件路径。
5. 选择Execute Packed Client File项， 会让Client Executable File Path after packing输入的可执行文件，自动生成hsb文件。

   注意事项!!!
   除了 upx加壳工具，使用其他的加壳工具(themida, asprotect 等..)时必须选择 execute client file 选项。

6. Parameters for Client Executable File to Execute 选项中添加 Execute Packed Client File 输入的可执行文件运行所需要的值。 （请参考图像）

7. Append LMP Data to Packed (or Original) Client File 选项是，如使用LMP功能时在发布的游戏客户端中添加LMP信息. 该LMP信息可以在没有加壳游戏客户端或加壳的游戏客户端两者都能添加。

8. 在HSB File Path输入框里设定要生成.hsb文件的Full Path. 点击[…]按钮，则生成文件的窗口将会被打开。（参考：假如设定在与执行文件相同的地方，则会出现可能与客户端文件一同分发的警告窗口。这是正确的消息窗口。）

9. 所有的设定结束之后，点击Generate按钮。进行的情况将被显示在Progress，结束时会提示正常结束的消息窗口。

✓ .hsb文件生成之后，复制到服务器可以loading的指定路径上。

❖ 与服务器相关的准备事项。

✓ 需要\Lib\ AntiCpXSvr.lib库文件。复制在恰当的位置，然后添加在游戏服务器Project里。

✓ 需要\Bin\AntiCrack\AntiCpXSvr.dll文件。复制在游戏服务器的执行路径里

✓ 需要\Bin\AntiCrack\HSPub.key文件。复制在游戏服务器的执行路径里(与生成的.hsb文件相同的位置)

✓ 需要\Bin\AntiCrack\3n.mhe文件。复制在游戏服务器的执行路径里(与生成的.hsb文件相同的位置)

✓ 需要\Bin\AntiCrack\hshield.dat文件。复制在游戏服务器的执行路径里(与生成的.hsb文件相同的位置)

☞ 参考： 3n.mhe 与 hshield.dat 文件的复制过程不是必须的，但是为了HackShield版本管理而推荐复制。

(服务器中hsb存在的位置上，上传客户端中使用的3n.mhe与hshield.dat。致使服务器上的3n.mhe与hshield.dat文件不让与版本不对的客户端进行连接)

- ✓ 需要\Include\ AntiCpXSvr.h 头文件。复制在适当的位置上.

❖ 与客户端相关的准备事项。

- ✓ 需要\Lib\HShield.lib库文件. 复制在适当的位置上，然后添加在游戏客户端Project里.
- ✓ 需要\Include\HShield.h 头文件. 复制在适当的位置上.

5 服务器 运用

❖ 将之前复制的 AntiCpXSvr.h 头文件Include到Server Project里的适当的位置上.

```
#include "AntiCpXSvr.h"
```

❖ AHNHS_TRANS_BUFFER的理解

在服务器上使用的_AhnHS_MakeRequest函数与在客户端上使用的 _AhnHS_MakeResponse 函数中将 AHNHS_TRANS_BUFFER 结构体作为buffer来显示结果。Buffer的结构如下。实际生成的消息将被保存在 byBuffer 数组里，相关数组的长度将被输出在 nLength里.

```
typedef struct _AHNHS_TRANS_BUFFER
{
    unsigned short nLength;
    unsigned char byBuffer[ANTICPX_TRANS_BUFFER_MAX/* 发送与接收包的最大值 */];
} AHNHS_TRANS_BUFFER, *PAHNHS_TRANS_BUFFER;
```

❖ 执行服务器之前对AntiCpXSvr进行初始化. (_AhnHS_CreateServerObject)

```
// ① 在游戏服务器被初始化的部分调用 _AhnHS_CreateServerObject.
void GameServer_OnInitialize()
{
    ....
    AHNHS_SERVER_HANDLE hServer = NULL;

    hServer = _AhnHS_CreateServerObject ( g_szHsbFilePath ); // .hsb 文件的路径

    if ( hServer == ANTICPX_INVALID_HANDLE_VALUE )
    {
        printf ( "ERROR: _AhnHS_CreateServerObject\n" );
        return;
    }
    ....
}
```

① g_szHsbFilePath是通过 HSBGen.exe生成的. Hsb文件的整个路径.

❖ 客户端每当生成Session时生成Client Handle. (_AhnHS_CreateClientObject)

```
void GameServer_OnLogOn()
{
    ....

    AHNHS_CLIENT_HANDLE hClient = ANTICPX_INVALID_HANDLE_VALUE;
    unsigned long ulRet = ERROR_SUCCESS;
    AHNHS_TRANS_BUFFER stRequestBuf;

    // 利用相关服务器的Handle 生成已连接用户的Handle.
    hClient = _AhnHS_CreateClientObject ( hServer );

    assert ( hClient != ANTICPX_INVALID_HANDLE_VALUE );
    if ( hClient == NULL )
    {
        printf ( "ERROR: _AhnHS_CreateClientObject\n" );
        return;
    }
}
```

❖ 在处理与客户端通信的thread函数中添加生成请求消息的处理模块。(_AhnHS_MakeRequest)

```
    // 利用相关用户的Handle制作请求消息

    // 各用户的请求消息是利用相关用户的Handle制作。
    ulRet = _AhnHS_MakeRequest ( hClient, &stRequestBuf );

    if ( ulRet != ERROR_SUCCESS )
    {
        printf ( "ERROR: _AhnHS_MakeRequest() == %Xh\n", ulRet );
        break;
    }

    // 通过Socket传送 stRequestBuf

    ....

    send ( sock, stRequestBuf.byBuffer, stRequestBuf.nLength, 0 );
```

❖ 从客户端应请求消息传送应答消息时，添加模块来检查应答消息的准确性。(_AhnHS_VerifyResponse)

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // 从客户端收信的应答buffer

    ....
```

```
    // 通过 Socket 收 stResponseBuf

    ....

    ulRet = _AhnHS_VerifyResponse ( hClient,
                                    stResponseBuf.byBuffer,
                                    stResponseBuf.nLength );

    if ( ulRet == ERROR_ANTICPXSVR_BAD_MESSAGE ||
         ulRet == ERROR_ANTICPXSVR_REPLY_ATTACK ||
         ulRet == ERROR_ANTICPXSVR_HSHIELD_FILE_ATTACK ||
         ulRet == ERROR_ANTICPXSVR_CLIENT_FILE_ATTACK ||
         ulRet == ERROR_ANTICPXSVR_MEMORY_ATTACK ||
         ulRet == ERROR_ANTICPXSVR_OLD_VERSION_CLIENT_EXPIRED ||
         ulRet == ERROR_ANTICPXSVR_NANOENGINE_FILE_ATTACK ||
        ulRet == ERROR_ANTICPXSVR_UNKNOWN_CLIENT ||
        ulRet == ERROR_ANTICPXSVR_INVALID_HACKSHIELD_VERSION ||
        ulRet == ERROR_ANTICPXSVR_INVALID_ENGINE_VERSION ||
        ulRet == ERROR_ANTICPXSVR_ABNORMAL_HACKSHIELD_STATUS    )


    {
        printf ( "ERROR: _AhnHS_VerifyResponse() == %Xh\n", ulRet );
    }

    printf( "SUCCESS: hClient = %d\n", hClient );

    ....
}
```

① 服务器通过调用_AhnHS_VerifyResponse函数，分析从客户端接收的版本应答消息。

② 如果正常(出现ERROR_SUCCESS)，则继续进行，不然就对错误进行处理并终止相关客户端的Socket，然后在连接List里删除。

❖ 从客户端传送应答消息时，检测该应答消息的函数 _AhnHS_VerifyResponse 可以改成_AhnHS_VerifyResponseEx 函数。（建议使用AhnHS_VerifyResponseEx 函数）

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // 从客户端收信的应答buffer
    ULONG ulLastError = 0;
    ....

    // 通过 Socket 收 stResponseBuf

    ....
```

```
    ulRet = _AhnHS_VerifyResponseEx ( hClient,
                                      stResponseBuf.byBuffer,
                                      stResponseBuf.nLength,
                                      &ulLastError);

    if ( ulRet == ANTICPX_RECOMMEND_CLOSE_SESSION )
    {
        //游戏客户端探测到黑客行为,
        //切断对游戏客户端的连接.
    }

    printf( "SUCCESS: hClient = %d\n", hClient );

    ....
}
```

① 服务器通过调用_AhnHS_VerifyResponse函数，分析从客户端接收的版本应答消息。

② 函数返回值 ANTICPX_RECOMMEND_KEEP_SESSION继续执行游戏，如果是 ANTICPX_RECOMMENT_CLOSE_SESSION就对错误进行处理并终止相关客户端的Socket，然后在连接List里删除。

❖ 客户端中止连接 ， 即Session终止的话 ， 通过_AhnHS_CloseClientHandle函数 ， 将Client Handle关掉。(_AhnHS_CloseClientHandle)

```
void GameClient_OnFinalize()
{
    ....
    _AhnHS_CloseClientHandle ( hClient );
    ....
}
```

❖ 终止游戏服务器时，通过_AhnHS_CloseServerHandle函数关掉Server Handle.(_AhnHS_CloseServerHandle)

```
void GameServer_OnFinalize()
{
....
_AhnHS_CloseServerHandle ( hServer );
....
}
```

6 客户端运用

与服务器通信的thread函数内添加对应请求消息的应答消息的处理模块. (_AhnHS_MakeRequest)

```
void GameClient_MakeResponse()
{
    AHNHS_TRANS_BUFFER stRequestBuf;   // 从服务器接到的请求消息
    AHNHS_TRANS_BUFFER stResponseBuf; // 向服务器传送的应答消息
```

```
    ....

    // 通过Socket接收 stRequestBuf

    ....

    ulRet = _AhnHS_MakeResponse ( stRequestBuf.byBuffer, stRequestBuf.nLength,
&stResponseBuf );

    if ( ulRet != ERROR_SUCCESS )
    {
        printf ( "ERROR: _AhnHS_MakeResponse() == %Xh\n", ulRet );
    }

    //通过 Socket向服务器传送stResponseBuf

    ....

    send ( sock, stResponseBuf.byBuffer, stResponseBuf.nLength, 0 );
}
```

✓ 从服务器收到请求消息，则制作相应的应答消息。

✓ 在客户端的的消息处理程序里添加请求消息的接收部分，然后调用_AhnHS_MakeResponse函数进行处理

7 嵌入时的注意事项

- 为了调试及客户支持，建议输出错误消息时将错误代码一同输出
- 发生错误时，只提示错误消息，有的时候游戏会继续进行，所以发生错误时应当将与客户端断开连接并请进行与相关客户端的Socket终止与在连接List中删除的工作。
- 如果对服务器的请求未做出应答，应当与客户端断开连接（推荐应答的时间：10秒内）
- 使用HSBGen.exe生成.hsb文件时，客户端的执行文件应是加壳之前的原文件。
- 测试时，hsb文件只认识用HSBGen.exe做过的游戏Client，测试时请使用该客户端

♉ 在Linux/Unix环境下区分大小文字，所以要正确的输入 HSPub.key文件名。

8 测试及分发

- 运行游戏客户端，并通过简单的测试判断扩充服务器联动的动作与否

- ✓ Case 1. 使用Edit打开客户端的执行文件并对特定的部分进行更改，然后运行游戏客户端并确认与服务器断开连接。

❖ 推荐分发时按照下列过程进行分发

- ✓ 运用及测试 ⇒ 进行Quality Assurance(Ahnlab) ⇒ 接收QA Report后确认修改事项，然后制作最终释放版本 ⇒ 分发