AhnLab HackShield

Quick Guide

Version 2.2

AhnLab HackShield for Online Game

Extended Server-side Detection

AhnLab

1 Preface

- This document is a guideline for the application of HackShield Pro SDK (or HackShield.) Please note that this document assumes that the application is written in C/C++ in Microsoft Visual 6.0 environment. It covers extended server-side detection.
- This document provides sample codes to help understand how it works. You can use it according to your company's policies.
- *"AhnLab HackShield Pro Programming Guide"* can be found in the Doc folder . This document provides the detailed information on AhnLab HackShield Pro and API functions. For more information, refer to this document.

2 Files for Extended Server Interoperability

- \Bin\Win\[Platform]\AntiCrack\AntiCpXSvr.dll: Dll file for extended server-side detection
- \Bin\Win\x86\AntiCrack\HSBGen.exe: hsb file creation tool
- \Bin\Win\x86\AntiCrack\HSPub.key : Authentication key file
- \Bin\Win\x86\HShield\3n.mhe: Heuristic engine version management file
- \Bin\Win\x86\HShield\hshield.dat: HackShield version management file
- \Include\AntiCpXSvr.h: Header file for extended server-side detection
- \Lib\Win\x86\AntiCpXSvr.lib: Library files for extended server-side detection (for Windows 32 Bit)
- \Lib\Win\x64\AntiCpXSvr.lib: Library files for extended server-side detection (for Windows 64 Bit)
- \Bin\Linux\x86\AntiCrack\ibanticpxsvr.so: Dynamic library files for extended server-side detection (for Linux 32 Bit)
- \Bin\Linux\x64\AntiCrack\ibanticpxsvr.so: Dynamic library files for extended server-side detection (for Linux 64 Bit)
- \Bin\Solaris\x86\AntiCrack\libanticpxsvr.so: Dynamic library files for extended server-side detection (for Solaris)
- \Lib\Linux\x86\AntiCrack\libanticpxsvr_st.a: Static library file for extended server-side detection (for Linux 32 Bit)
- \Lib\Solaris\x86\AntiCrack\libanticpxsvr_st.a: Static library file for extended server-side detection (for Solaris)

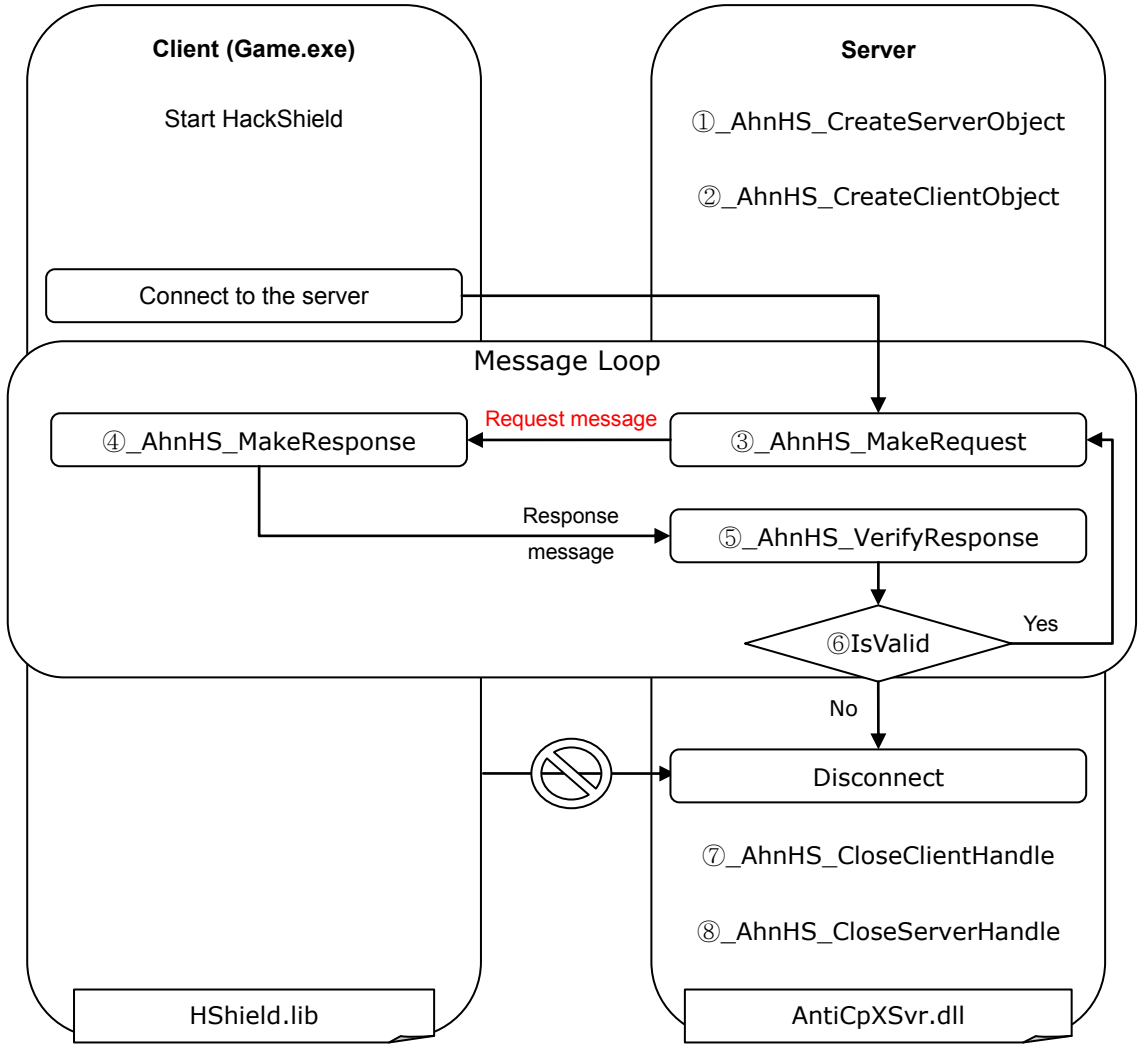
※ Files for Solaris support 32 bit x86 platform only.

3 Operating Principles

- HackShield extended server-side detection has been adopted from the existing communication structure between server and client. (There is no need to implement a separate communication module.)
- Check the version of the client in order to prevent unauthorized execution.
  - ✓ In case the version is not allowed, the game will not run unless the latest patch is downloaded.
- Check if there is any crack in the client executable file, memory, and HackShield module.
  - ✓ Regularly check the request and response according to the security policy.
  - ✓ If the response to the request is not valid, it will be considered as a hack attack and the client will be disconnected.

✓ In case the response is not arrived, it will be also considered as a hack attack and the client will be disconnected.

❖ The following chart shows the general operating principles.



① When the game server is first run, AHNHS_SERVER_HANDLE will be created by _AhnHS_CreateServerObject function. This handle shall be kept until the game server is terminated. (Note: The server handle is used to create a client handle in the next two steps.)

② Each time a client accesses the server, AHNHS_CLIENT_HANDLE will be created by _AhnHS_CreateClientObject function with the server handle being used as a parameter. This handle shall be maintained during the session in which the client is connected to the network. (Note: The client handle is used to create a request message in the Step 3 below.)

③ In order to monitor client manipulation, a request message is created and sent to client. .The request message is created by _AhnHS_MakeRequest function using a client handle as a parameter.

④ The client sends a response message to the server. The response message is created by _AhnHS_MakeResponse function.

⑤ Check whether the response message from the client is valid. The validity of the response message is verified by _AhnHS_VerifyResponse function.

⑥ In case an error code apart from ERROR_SUCCESS is returned by _AhnHS_VerifyResponse

function, the error shall be handled according to error code and the game client will be disconnected.

⑦ When the client is disconnected (with the session being terminated), the client handle created in the Step 2 will be closed.

⑧ When the server process is terminated, the server handle created in the Step 1 will be closed. (At this time, all client handles must be closed.)

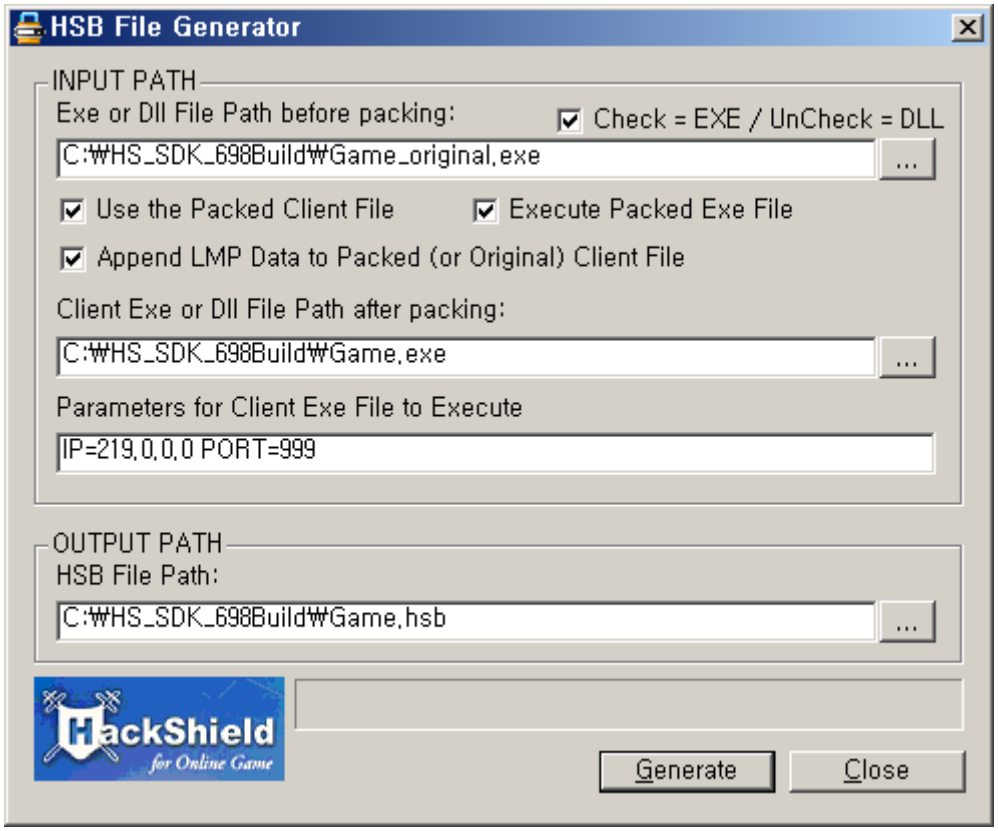
4 Getting Started

❖ Create .hsb file, using \Bin\Win\x86\AntiCrack\HSBGen.exe.

✓ Important!!! The client executable file must be the original file before it is packed.

✓ Important!!! If you want to distribute the client executable file with digital signature, sign the file digitally after running HSBGen tool.

✓ Create .hsb file using HSBGen.exe as follows:



1. Check the checkbox at the right side of **Exe or Dll File Path before packing** if the target file is an exe file, and uncheck it if the target file is a DLL file.
2. Specify the path of the original executable file which has not been packed in the **Exe or Dll File Path before packing** field. You can browse and select the file by clicking the **…** button.
3. In order to distribute the packed client executable file, enter the path of the original file in the first field.
4. Select **Use the Packed Client File**, and enter the path of the packed file in the **Client Exe or Dll File Path after packing** field.

5. Select **Execute Packed Client File**. It automatically runs executable files entered in Client Executable File Path after packing and decides whether to create a hsb file.

   Caution
   If you are using packing programs other than upx, such as themida and asprotect, **Execute Packed Client File** must be selected.

6. Enter the value in **Parameters for Client Executable File to Execute** to run the client file.
7. Select **Append LMP Data to Packed (or Original) Client File** to add LMP information to the game client file. The LMP information can be added to the unpacked original executable file or the packed client executable file.
8. Specify the full path of .hsb file in the **HSB File Path** field. Click the [**…**] button. (Note: If you specify the same folder as that of the executable file, a warning message that the client file will be also distributed will pop up. This is a reminder for the user.
9. After completing all settings, click **Generate** button. The progress will be displayed on the progress bar. After completion, a completion message will be displayed.

✓ Copy .hsb file to a path where the file can be loaded in the server.

❖ Preparation for Server
- ✓ \Lib\Win\x86\AntiCpXSvr.lib file is required. Copy and add it to the game server project.
- ✓ \Bin\Win\x86\AntiCrack\AntiCpXSvr.dll file is required. Copy it to the execution path of the server.
- ✓ \Bin\AntiCrack\HSPub.key file is required. Copy it to the execution path of the server. (same as the .hsb file path.)
- ✓ \Bin\Win\x86\AntiCrack\3n.mhe file is required. Copy it to the execution path of the server. (same as the .hsb file path.)
- ✓ \Bin\Win\x86\HShield\hshield.dat file is required. Copy it to the execution path of the server. (same as the .hsb file path.)
  - ➢ Note: Copying 3n.mhe and hshield.dat files is not required, but it is recommended for HackShield version management.
    (Upload 3n.mhe and hshield.dat files used by the client to the folder where the hsb file is located to prevent the client with the wrong 3n.mhe and hshield.dat version from connecting.)
- ✓ \Include\AntiCpXSvr.h header file is required. Copy the file to a proper folder.

❖ Preparation for Client
- ✓ \Lib\Win\x86\HShield.lib library file is required. Copy it to a proper path and add to the game client project.
- ✓ \Include\HShield.h header file is required. Copy the file to a proper folder.

5 Server Application

❖ Include the AntiCpXSvr.h file to the server project.

```
#include “AntiCpXSvr.h”
```

❖ What is AHNHS_TRANS_BUFFER?
_AhnHS_MakeRequest function and _AhnHS_MakeResponse function receive AHNHS_TRANS_BUFFER structure in the buffer and display the result. The buffer structure is as

shown below. The message created is saved in byBuffer arrangement, and the length of the arrangement will be displayed in nLength.

```
typedef struct _AHNHS_TRANS_BUFFER
{
    unsigned short nLength;
    unsigned char byBuffer[ANTICPX_TRANS_BUFFER_MAX/* Maximum packet size */];
} AHNHS_TRANS_BUFFER, *PAHNHS_TRANS_BUFFER;
```

❖ Before executing the server, initialize AntiCpXSvr. (_AhnHS_CreateServerObject)

```
// ① Call _AhnHS_CreateServerObject in the part where the game server is initialized.
void GameServer_OnInitialize()
{
    ….
    AHNHS_SERVER_HANDLE hServer = NULL;

    hServer = _AhnHS_CreateServerObject ( g_szHsbFilePath ); // .hsb file path

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

① g_szHsbFilePath is the full path for . hsb file created through HSBGen.exe.

❖ Each time the client generates a session, a client handle will be created. (_AhnHS_CreateClientObject)

```
void GameServer_OnLogOn()
{
    ….

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

    // Create a handle of the connected user using the server handle
    hClient = _AhnHS_CreateClientObject ( hServer );

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

❖ Add a request message creation module in the thread function which handles the communication with the client. (_AhnHS_MakeRequest)

```
// Write a request message using the user's handle.
// Write each user's request message using the user handle.
ulRet = _AhnHS_MakeRequest ( hClient, &stRequestBuf );

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

// Send stRequestBuf through the socket.

....

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

❖ Add a module to verify the response message from the client. (_AhnHS_VerifyResponse)

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // Response buffer received from the client

    ....

    // Receive stResponseBuf through the socket.

    ....

    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_REPLY_ATTACK ||
         ulRet == ERROR_ANTICPXSVR_INVALID_ENGINE_VERSION ||
         ulRet == ERROR_ANTICPXSVR_ABNORMAL_HACKSHIELD_STATUS ||
         ulRet == ERROR_ANTICPXSVR_DETECT_CALLBACK_IS_NOTIFIED )
         )
    {
         printf ( "ERROR: _AhnHS_VerifyResponse() = %X\n", ulRet );
    }

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

```
    ....
}
```

① The server calls _AhnHS_VerifyResponse function and analyzes the version response message received from the client.

② If the message is not the one defined by “if” statement in the example, continue the process as it is normal. Otherwise, terminate the client socket and remove the client from the connection list.

❖ Instead of the _AhnHS_VerifyResponse function which is used to verify the response, you can use the _AhnHS_VerifyResponseEx function mentioned below.

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // Response buffer received from the client
    ULONG ulLastError = 0;
    ....

    // Receive stResponseBuf through the socket.

    ....

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

    if ( ulRet == ANTICPX_RECOMMAND_CLOSE_SESSION )
    {
        // A hack attack has been detected in the game client,
        // so the connection to the game client will be blocked.
    }

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

    ....
}
```

① The server calls the _AhnHS_VerifyResponseEx function and analyzes the version response message received from the client.

② If the returned value is ANTICPX_RECOMMEND_KEEP_SESSION, it is normal and the session will continue. If the returned value is ANTICPX_RECOMMENT_CLOSE_SESSION, this is processed as an error, and the client socket will be terminated and removed from the connection list.

③ Internally, the _AhnHS_VerifyResponseEx() function calls the _AhnHS_VerifyResponse function. If the returned value is one of the below, ANTICPX_RECOMMAND_CLOSE_SESSION is returned.

- ERROR_ANTICPXSVR_BAD_MESSAGE
- ERROR_ANTICPXSVR_REPLY_ATTACK
- ERROR_ANTICPXSVR_UNKNOWN_CLIENT
- ERROR_ANTICPXSVR_HSHIELD_FILE_ATTACK

- ERROR_ANTICPXSVR_CLIENT_FILE_ATTACK
- ERROR_ANTICPXSVR_MEMORY_ATTACK
- ERROR_ANTICPXSVR_OLD_VERSION_CLIENT_EXPIRED
- ERROR_ANTICPXSVR_NANOENGINE_FILE_ATTACK
- ERROR_ANTICPXSVR_INVALID_HACKSHIELD_VERSION
- ERROR_ANTICPXSVR_INVALID_ENGINE_VERSION
- ERROR_ANTICPXSVR_VERIFY_EXCEPTION
- ERROR_ANTICPXSVR_INVALID_PARAMETER
- ERROR_ANTICPXSVR_ABNORMAL_HACKSHIELD_STATUS
- ERROR_ANTICPXSVR_DETECT_CALLBACK_IS_NOTIFIED

❖ If the client is disconnected and the session is terminated, the client handle will be closed through _AhnHS_CloseClientHandle function. (_AhnHS_CloseClientHandle)

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

❖ When terminating the game server, the server handle will be closed through _AhnHS_CloseServerHandle function. (_AhnHS_CloseServerHandle)

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

6 Application to Client

❖ Add a response message handling module in the thread function which is responsible for communication with the server. (_AhnHS_MakeRequest)

```
void GameClient_MakeResponse()
{
    AHNHS_TRANS_BUFFER stRequestBuf;          // Request message received from the server
    AHNHS_TRANS_BUFFER stResponseBuf; // Response message to be sent to the server

    ....

    // Receive stRequestBuf through the socket.

    ....

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

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

    // Send to stResponseBuf server through the socket.

    ....

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

- ✓ After a request message is received from the server, the client writes a response message.
- ✓ Add a request message receiving part to the message handling routine of the client, and call _AhnHS_MakeResponse function.

7 Caution

- ❖ It is recommended to display error messages with error codes for debugging and customer support.
- ❖ Users may ignore the error message and continue playing the game. It is highly recommended to forcibly terminate the game client, and the client socket, and remove the client from the connection list, as well as display an error message.
- ❖ In case there is no response to the server request, the connection shall be terminated. (Recommended connection timeout: 10 seconds)
- ❖ When creating .hsb file using HSBGen.exe, the client execution file must be the original file before it is packed.
- ❖ Linux/Unix environment is case-sensitive so HSPub.key file names must be correctly entered.

8 Test and Distribution

- ❖ Verify whether the server-side detection works normally by running the game client and conducting some tests:
  - ✓ Case 1. Open the client executable file with an editor program, and make changes. Then, access the server and check whether the connection is disconnected.
- ❖ It is recommended to distribute as below:
  - ✓ Build and test ⇒ Quality assurance (by AhnLab) ⇒ Receive QA report, review, and write final version for distribution ⇒ Distribute