AhnLab HackShield

Quick Guide

V 2.0

拡張サーバー連動機能

1. はじめに

- ❖ 本書は HackShield Pro SDK 適用のためのガイドラインです。Microsoft Visual C++ 6.0 環境において C/C++ 言語でプログラム作成することを基準とし、HackShield の拡張サーバー連動機能について説明します。
- ❖ 初めて HackShield をプログラムコードに適用する際の概略説明および簡略化したサンプルコードが収められています。コード実装に関する詳細についてはゲーム開発会社のポリシーに沿って進行して下さい。
- ❖ ¥Doc フォルダ内にある「AhnLab HackShield Pro プログラミングガイド」は HackShield Pro の全体構造や機能、API 関数の使用法についてのプログラミングガイドです。項目別の詳細についてはプログラミングガイドを参照して下さい。
- ❖ 本書に記載されているすべての製品名は該当各社の標章、商標または登録商標です。

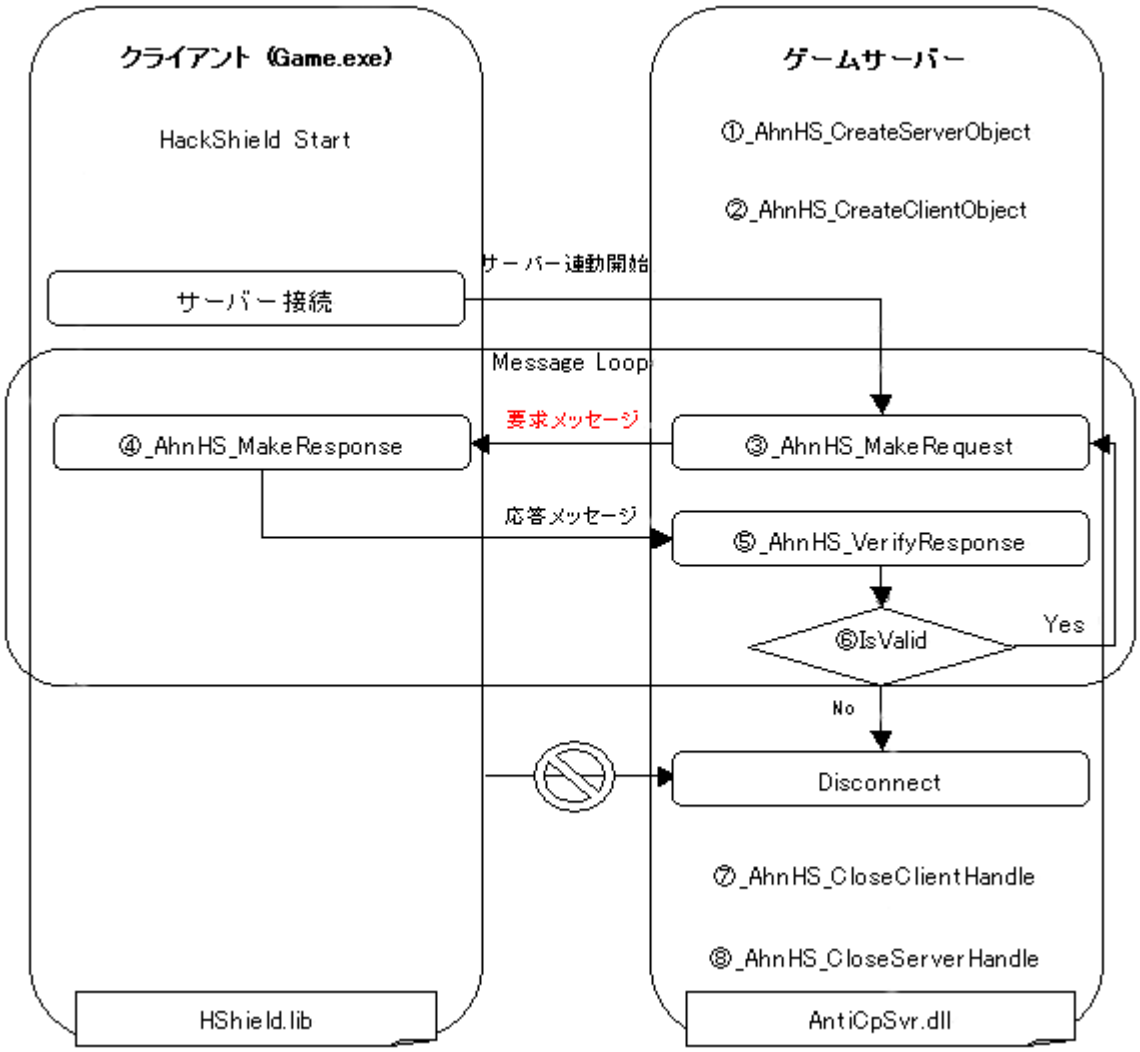
2. HackShield Pro 拡張サーバー連動機能の構成ファイル

- ❖ ¥Bin¥AntiCrack¥AntiCpXSvr.dll : 拡張サーバー連動 dll ファイル
- ❖ ¥Bin¥AntiCrack¥HSBGen.exe : .hsb ファイル生成ツール
- ❖ ¥Include¥AntiCpXSvr.h : 拡張サーバー連動インクルードファイル
- ❖ ¥Lib¥AntiCpXSvr.lib : 拡張サーバー連動ライブラリファイル

3. 動作構造

- ❖ HackShield の拡張サーバー連動通信は既存のゲームで実装されている通信機能を使用します。(別途通信モジュールを実装する必要はありません。)
- ❖ クライアントのバージョンをチェックして不正実行を防止します。
  - ✓ 許可していないバージョンであれば接続を切り、最新のパッチを適用せずにゲームを続けることを防止します。
- ❖ クライアントの実行ファイル、メモリ、HackShield モジュールに対するクラック有無を判断します。
  - ✓ ポリシーに沿って要求メッセージとこれに対する応答メッセージを定期的に確認します。
  - ✓ 要求メッセージに対する応答メッセージが有効でない場合、ハッキングと判断して接続を切ります。
  - ✓ 応答メッセージが転送されない場合もハッキングと判断して接続を切ります。

❖ 全体的な動作構造は下図のとおりです。



① ゲームサーバー起動時に _AhnHS_CreateServerObject 関数から AHNHS_SERVER_HANDLE を生成します。このハンドルはゲームサーバーが終了するまで維持する必要があります。(参考 : サーバーハンドルは次の段階でクライアントハンドルを作成する際に使用します。)

② 各クライアントが接続する度に _AhnHS_CreateClientObject 関数とサーバーハンドルをパラメータとして与え、AHNHS_CLIENT_HANDLE を生成します。このハンドルはクライアントのネットワークが接続しているセッションの間維持されます。(参考 : クライアントハンドルは次の段階で要求メッセージを生成する際に使用します。)

③ クライアントの偽造/改ざん有無を監視するために要求メッセージを生成して転送します。要求メッセージは _AhnHS_MakeRequest 関数から生成し、クライアントハンドルをパラメータとして使用します。

④ クライアントはサーバーの要求メッセージに適切な応答メッセージを生成します。応答メッセージは _AhnHS_MakeResponse 関数から生成します。

⑤ クライアントの応答メッセージが有効であるかスキャンします。応答メッセージの有効性スキャンは _AhnHS_VerifyResponse 関数で行います。

⑥ _AhnHS_VerifyResponse 関数で ERROR_SUCCESS 以外のエラーコードが発生した場合にはエラーコードに従って適切なエラー処理を行った後、ゲームクライアントの接続を中断します。

⑦ クライアントの接続を切る時（セッションが終了する時）には第 2 段階で生成したクライアントハンドルを閉じます。

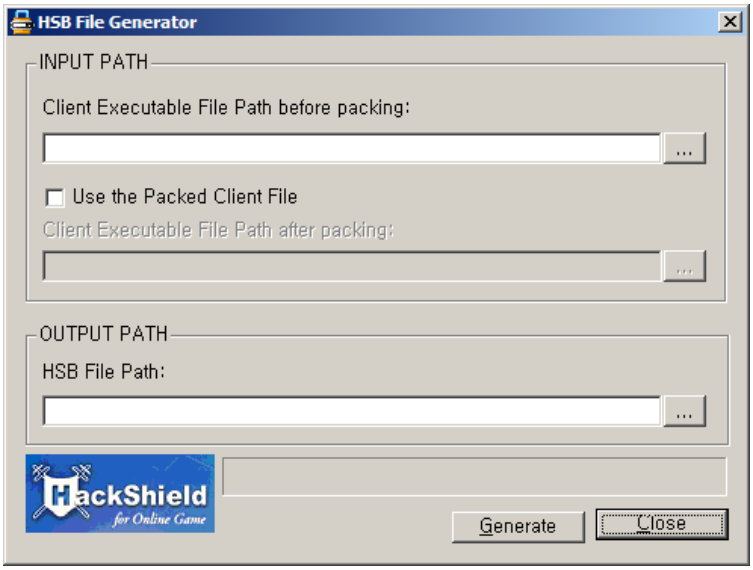
⑧ サーバープロセスを終了する時には第 1 段階で生成したサーバーハンドルを閉じます。(この時クライアントハンドルがすべて閉じた状態である必要があります。)

4. 事前準備事項

❖ ¥Bin¥AntiCrack¥HSBGen.exe を利用して AntiCpX.hsb ファイルを生成します。

✓ 注意事項!!! クライアント実行ファイルはパッキング前の原本リリースファイルである必要があります。

✓ HSBGen.exe で AntiCpX.hsb ファイルを生成する方法は次のとおりです。



1. **Client Executable File Path before packing** 入力ボックスに原本の実行ファイルパスを設定します。[…] ボタンをクリックしてファイルを選択します。
2. クライアント実行ファイルをパッキングして配布する予定の場合、第 1 段階で説明したフィールドにはパッキングする前のファイルパスを入力します。
3. **[Use the Packed Client File]** チェック ボックスをオンにして **Client Executable File Path after packing** 入力ボックスへパッキング後のファイルパスを入力します。
4. **HSB File Path** 入力ボックスに **AntiCpX.hsb** ファイルが生成されるフルパスを設定します。[…] ボタンをクリックするとファイルを生成するウィンドウが開きます。(参考 : 実行ファイルと同じ場所に設定すると、クライアントファイルと一緒に配布されることがあるという警告メッセージが表示されますが、これは正しいメッセージです。)
5. すべての設定が完了したら **[Generate]** ボタンをクリックします。進行状況がプログレスバーに表示され、設定の完了とともに正常に完了したことを通知するメッセージが表示されます。

✓ AntiCpX.hsb ファイルの生成が完了したら、サーバーからロード可能なパスにコピーします。

❖ サーバー関連準備事項
- ✓ ¥Lib¥ AntiCpXSvr.lib ライブラリファイルを適当な場所にコピーし、ゲームサーバープロジェクトに追加します。
- ✓ ¥Bin¥AntiCrack¥AntiCpXSvr.dll ファイルをゲームサーバーの実行パスにコピーします。
- ✓ ¥Include¥ AntiCpXSvr.h ヘッダーファイルを適当な場所にコピーします。

❖ クライアント関連準備事項
- ✓ ¥Lib¥HShield.lib ライブラリファイルを適当な場所にコピーし、ゲームクライアントプロジェクトに追加します。
- ✓ ¥Include¥HShield.h ヘッダーファイルを適当な場所にコピーします。

5. サーバー適用

❖ サーバープロジェクトに先ほどコピーした AntiCpXSvr.h ファイルをインクルードします。

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

❖ AHNHS_TRANS_BUFFER の理解

サーバーで使用する _AhnHS_MakeRequest 関数とクライアントで使用する _AhnHS_MakeResponse 関数では AHNHS_TRANS_BUFFER 構造体をバッファに書き込み、結果を出力します。バッファの構造は以下のとおりです。生成されたメッセージは byBuffer 配列に保存され、この配列の長さは nLength に出力されます。

```
typedef struct _AHNHS_TRANS_BUFFER
{
    unsigned short nLength;
    unsigned char byBuffer[ANTICPX_TRANS_BUFFER_MAX/*送受信パケットの最大サイズ*/];
} AHNHS_TRANS_BUFFER, *PAHNHS_TRANS_BUFFER;
```

❖ サーバーを実行する前に AntiCpXSvr の初期化を実行します。(_AhnHS_CreateServerObject)

```
//①ゲームサーバーが初期化される部分に _AhnHS_CreateServerObject を呼び出します。
void GameServer_OnInitialize()
{
    ....
    AHNHS_SERVER_HANDLE hServer = NULL;

    hServer = _AhnHS_CreateServerObject ( g_szHsbFilePath );     //.hsb ファイルパス

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

① g_szHsbFilePath は HSBGen.exe で生成した .hsb ファイルのフルパスです。

❖ クライアントがセッションを生成する度にクライアントハンドルを生成します。(_AhnHS_CreateClientObject)

```
void GameServer_OnLogOn()
```

```
{
    ....

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

    //このサーバーのハンドルを利用して接続したユーザーのハンドルを生成
    hClient = _AhnHS_CreateClientObject ( hServer );

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

❖ クライアントとの通信を処理するスレッド関数に要求メッセージを生成する処理モジュールを追加します。(_AhnHS_MakeRequest)

```
    //ユーザーのハンドルを利用して要求メッセージを作成
    //各ユーザーの要求メッセージは各ユーザーのハンドルを利用して作成
    ulRet = _AhnHS_MakeRequest ( hClient,&stRequestBuf );

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

    // Socket から stRequestBuf 送信

    ....

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

❖ クライアントから要求メッセージに対する応答メッセージが転送された時にメッセージを検証するモジュールを追加します。(_AhnHS_VerifyResponse)

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf;     //クライアントから受信した応答バッファ

    ....

    // Socket から stResponseBuf 受信

    ....
```

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

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

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

    ....
}
```

① 서버는 _AhnHS_VerifyResponse 함수를 호출하여 클라이언트로부터 받은 버전 응답 메시지를 분석합니다.

② 정상(ERROR_SUCCESS)이면 연결을 유지하고, 정상이 아니면 오류 처리를 하고 이 클라이언트의 소켓을 종료하고 연결 목록에서 제거합니다.

❖ 클라이언트가 연결을 중단하고 세션이 종료되면 _AhnHS_CloseClientHandle 함수가 클라이언트 핸들을 닫습니다.(_AhnHS_CloseClientHandle)

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

❖ 게임 서버를 종료할 때 _AhnHS_CloseServerHandle 함수가 서버 핸들을 닫습니다.(_AhnHS_CloseServerHandle)

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

6. 클라이언트 적용

❖ 서버와의 통신을 처리하는 스레드 함수에 요청 메시지에 대한 응답 메시지 처리 모듈을 추가합니다.(_AhnHS_MakeRequest)

```
void GameClient_MakeResponse()
{
    AHNHS_TRANS_BUFFER stRequestBuf;      //서버에서 수신한 요청 메시지
    AHNHS_TRANS_BUFFER stResponseBuf;     //서버로 전송할 응답 메시지

    ....
```

```
    // Socket から stRequestBuf 受信

    ....

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

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

    // Socket から stResponseBuf サーバーに転送

    ....

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

- ✓ サーバーから要求メッセージを受信したらそれに対する応答メッセージを作成します。
- ✓ クライアントのメッセージ処理ルーチンに要求メッセージ受信の部分を追加し、_AhnHS_MakeResponse 関数を呼び出して処理します。

7. 適用時注意事項

- ❖ デバッグおよびカスタマーサポートのためにエラーメッセージを出力する場合はエラーコードを併せて出力することをお勧めします。
- ❖ エラー発生時にはメッセージを出力するだけではなく、必ずクライアントとの接続を終了し、該当クライアントのソケット終了と接続リストから削除する作業を行って下さい。
- ❖ サーバーの要求に対する応答がない場合は接続を終了するように構成する必要があります。(推奨応答時間 : 10 秒)
- ❖ HSBGen.exe で .hsb ファイルを生成する際、クライアントの実行ファイルはパッキング前の原本ファイルである必要があります。

8. テストおよび配布

- ❖ ゲームクライアントを実行して簡単なテストを行い、拡張サーバー連動の動作可否を確認します。
  - ✓ Case 1. クライアントの実行ファイルをエディターで開いて一部を改ざんし、実行します。サーバーに接続した時に接続が切れることを確認します。
- ❖ 配布時には、次の過程に従って配布して下さい。
  - ✓ 適用およびテスト ⇒ Quality Assurance 進行 (Ahnlab) ⇒ QA Report を受け取り、修正事項を確認した上、最終配布バージョンを作成 ⇒ 配布