AhnLab

HackShield Pro

for Online Game

Quick Guide

V 2.0

확장 서버 연동 기능

안철수연구소

1 시작하기 전에

- 이 문서는 HackShield Pro(이하 핵쉴드) SDK 적용을 위한 가이드라인입니다. Microsoft Visual C++ 6.0 환경에서 C/C++ 언어로 작성하는 것을 기준으로 작성되었습니다. 핵쉴드의 확장 서버 연동 기능에 대한 부분을 다룹니다.
- 처음 핵쉴드 코드 적용을 돕기 위하여 대략적인 설명과 간략한 샘플 코드를 다룹니다. 세부 구현은 각 게임 개발사의 정책에 맞추어 진행하십시오.
- \Doc 폴더 안에 *"AhnLab HackShield Pro 프로그래밍 가이드"* 문서가 제공됩니다. 이 문서는 HackShield Pro의 전체 구조와 기능 및 API 함수 사용법을 설명한 프로그래밍 가이드 입니다. 항목별 자세한 사항은 이 문서를 참고하십시오.

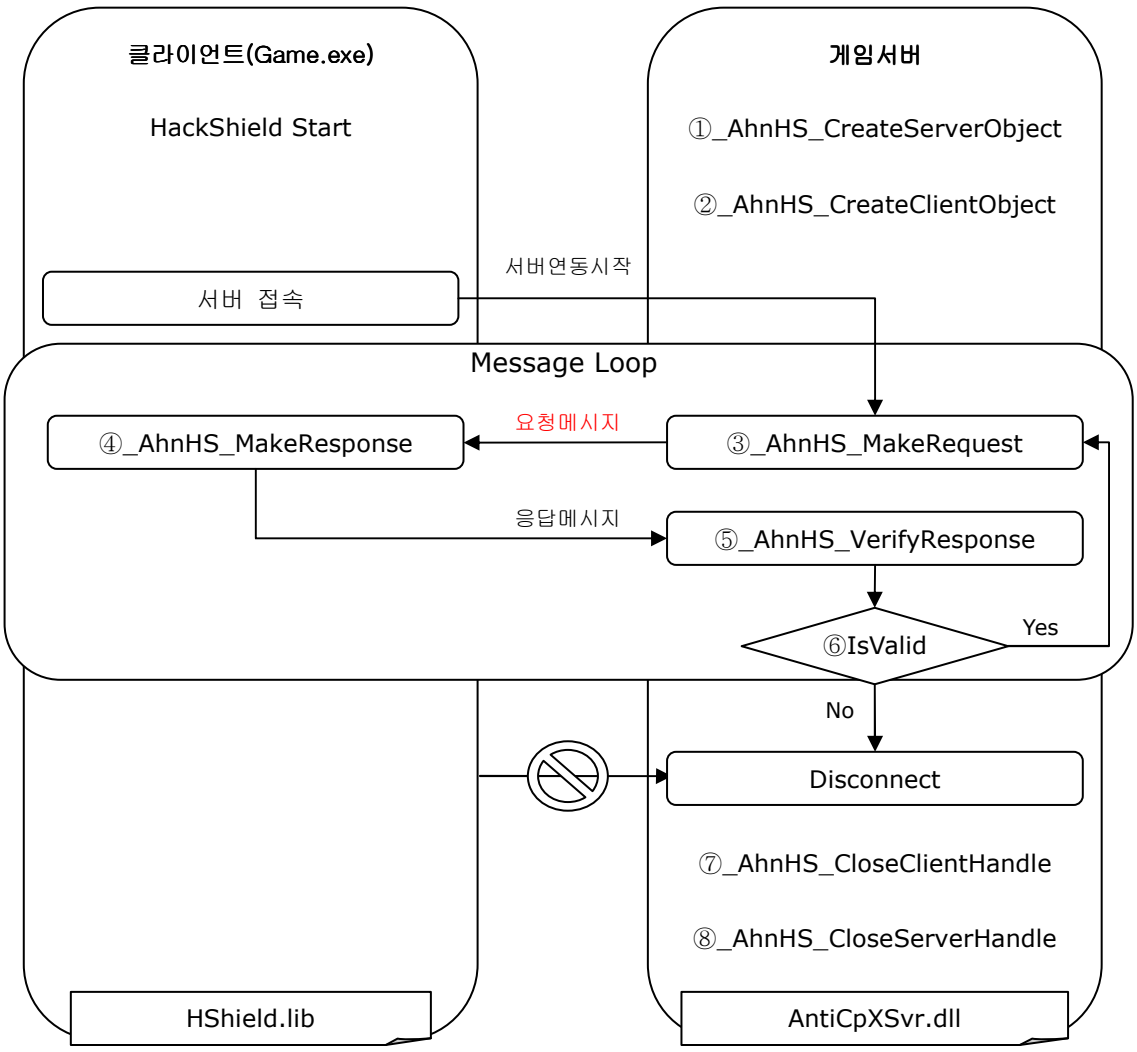
2 HackShield Pro 확장 서버 연동 기능의 구성 파일

- \Bin\AntiCrack\AntiCpXSvr.dll : 확장 서버 연동 dll 파일.
- \Bin\AntiCrack\HSBGen.exe : .hsb 파일 생성 툴
- \Bin\AntiCrack\HSPub.key : 서버연동 인증 키 파일
- \Bin\HShield\3N.mhe : 휴리스틱 엔진 버전 관리 파일
- \Bin\HShield\HShield.dat : 핵쉴드 버전 관리 파일
- \Include\AntiCpXSvr.h : 확장 서버 연동 헤더 파일
- \Lib\AntiCpXSvr.lib : 확장 서버 연동 라이브러리 파일 (Windows 용)
- \Bin\AntiCrack\Linux\libanticpxsvr.so : 확장 서버 연동 동적 라이브러리 파일 (Linux용)
- \Bin\AntiCrack\Solaris\libanticpxsvr.so :확장 서버 연동 동적 라이브러리 파일 (Solaris용)
- \Lib\Linux\libanticpxsvr_st.a : 확장 서버 연동 정적 라이브러리 파일 (Linux용)
- \Lib\Solaris\libanticpxsvr_st.a : 확장 서버 연동 정적 라이브러리 파일 (Solaris용)

※ Linux용 및 Solaris용 파일은 32bit x86 플랫폼만을 지원하고 있습니다.

3 동작 구조

- 핵쉴드 확장 서버 연동 통신 구조는 기존의 게임에 구현된 통신 구조를 그대로 사용합니다 (별도의 통신 모듈을 구현할 필요가 없습니다).
- 클라이언트의 버전을 체크하여 부정 실행을 방지합니다.
  - ✓ 허용하지 않는 버전이라면 연결을 끊어서 최신 패치를 받지 않고 게임을 진행하는 것을 방지합니다.
- 클라이언트의 실행 파일, 메모리, 핵쉴드 모듈에 대한 크랙 여부를 판단합니다.
  - ✓ 정책에 따라 일정한 시간 간격으로 요구 메시지와 이에 대한 응답 메시지를 확인합니다.
  - ✓ 요구 메시지에 대한 응답 메시지가 유효하지 않다면 해킹으로 판단하여 접속을 끊습니다.
  - ✓ 응답 메시지가 전송되지 않는 경우도 해킹으로 판단하여 접속을 끊습니다.
- 전체적인 동작 구조는 다음 그림과 같습니다.



① 게임 서버가 최초 실행될 때 _AhnHS_CreateServerObject 함수를 통해 AHNHS_SERVER_HANDLE을 생성합니다. 이 핸들은 게임 서버가 종료되기 전까지 유지되어야 합니다. (참고: 서버 핸들은 다음 2 단계의 클라이언트 핸들을 만드는데 사용됩니다.)

② 각 클라이언트가 접속 될 때 마다 _AhnHS_CreateClientObject 함수와 서버 핸들을 파라미터로 취하여 AHNHS_CLIENT_HANDLE을 생성합니다. 이 핸들은 클라이언트 네트워크 접속이 유지하는 세션 동안 유지합니다. (참고: 클라이언트 핸들은 다음 3 단계의 요청 메시지를 만드는데 사용됩니다.)

③ 클라이언트의 위변조 여부를 감시하기 위해 요청 메시지를 생성하여 전송합니다. 요청 메시지는 _AhnHS_MakeRequest 함수를 사용하여 생성하며 클라이언트 핸들을 파라미터로 사용합니다.

④ 클라이언트는 서버의 요청 메시지에 적절한 응답 메시지를 생성합니다. 응답 메시지는 _AhnHS_MakeResponse 함수를 통해 생성합니다.

⑤ 클라이언트의 응답 메시지가 유효한지 검사합니다. 응답 메시지의 유효성 검사는 _AhnHS_VerifyResponse 함수를 통해 검사합니다.

⑥ _AhnHS_VerifyResponse 함수에서 ERROR_SUCCESS 값 이외의 오류 코드가 발생했을 경우 오류 코드에 따라 적절하게 에러 처리를 수행한 후 게임클라이언트 접속을 중단합니다.

⑦ 클라이언트 접속을 끊을 때 (세션이 종료될 때) 2 단계에서 생성한 클라이언트 핸들을 닫습니다.

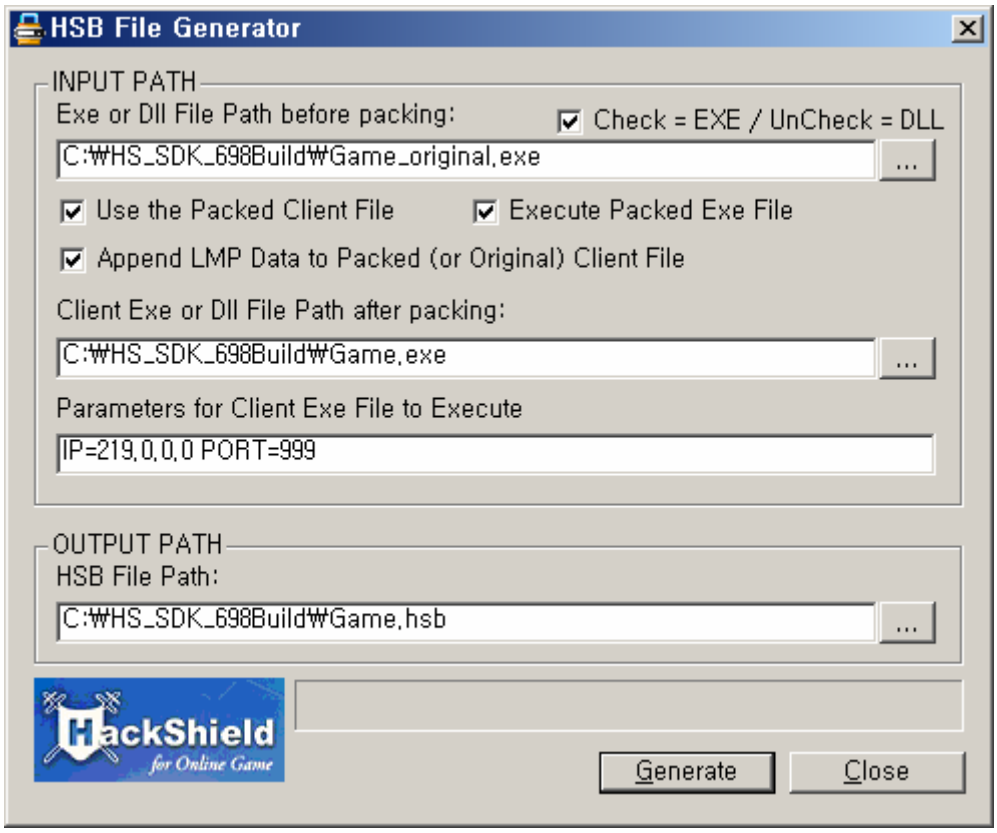
⑧ 서버 프로세스가 종료될 때 1 단계에서 생성한 서버핸들을 닫습니다. (이때 클라이언트 핸들이 모두 Close된 상태이어야 합니다.)

4 적용 전 준비사항

❖ \Bin\AntiCrack\HSBGen.exe를 이용하여 .hsb파일을 생성합니다.

✓ 주의 사항!!! 클라이언트 실행 파일은 패킹되기 이전의 원본 릴리즈 파일이어야 합니다.

✓ HSBGen.exe를 통해 .hsb 파일을 생성하는 방법은 다음과 같습니다.



1. 대상 파일이 exe일 경우는 Check , DLL일 경우는 UnCheck를 합니다.
2. **Exe or Dll File Path before packing** 입력 상자에 패킹되지 않은 원본 실행 파일의 경로를 설정합니다. […] 버튼을 눌러 파일을 선택합니다.
3. 패킹된 클라이언트 실행파일을 배포할 계획이라면, 앞서 설명한 첫 번째 필드에 패킹되기 이전의 파일 경로를 입력합니다.
4. **Use the Packed Client File** 항목을 선택하고 **Client Exe or Dll File Path after packing** 입력 상자에 패킹된 파일 경로를 입력합니다.

5. **Execute Packed Client File**은 Client Executable File Path after packing에서 입력한 실행파일을 자동으로 실행시킨후 hsb파일을 생성할것인지를 결정합니다.

   주의 사항!!!
   패킹 프로그램이 upx 이외의 다른 패커(themida, asprotect 등..)로 패킹된 경우는 execute client file 항목을 선택해야만 합니다.

6. Parameters for Client Executable File to Execute 항목은 Execute Packed Client File 에서 입력된 실행 파일을 실행시에 필요로하는 값들이 있다면 해당 값들을 본 항목에 입력하시기 바랍니다.
7. Append LMP Data to Packed (or Original) Client File 항목은 LMP 기능을 이용하고자 할 경우, 배포되는 게임 클라이언트 파일에 LMP 정보를 추가합니다. 해당 LMP 정보는 패킹되지 않은 원본 실행 파일에 추가하거나 패키된 클라이언트 실행 파일 둘 다 가능합니다.
8. **HSB File Path** 입력 상자에 .hsb 파일이 생성될 Full Path를 설정합니다. […] 버튼을 누르면 파일을 생성하는 창이 열립니다. (참고: 실행 파일과 같은 곳을 설정하면, 클라이언트 파일과 같이 배포될 수 있다는 경고 창이 나타납니다. 이것은 사용자에게 환기시키기 위한 목적으로 나타나는 정상 메시지입니다.)
9. 모든 설정이 완료되면 **Generate** 버튼을 누릅니다. 진행 상황이 프로그레스 막대에 표시되며, 완료와 함께 정상적으로 완료되었다는 메시지가 나타납니다.

✓ .hsb 파일 생성이 완료되면 서버에서 로딩할 수 있는 특정 경로에 복사합니다.

❖ 서버 관련 준비사항

✓ \Lib\ AntiCpXSvr.lib 라이브러리 파일이 필요합니다. 적당한 위치에 복사한 다음 게임 서버 프로젝트에 추가합니다.

✓ \Bin\AntiCrack\AntiCpXSvr.dll 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다.

✓ \Bin\AntiCrack\HSPub.key 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다. (생성될 .hsb 파일과 동일한 위치)

✓ \Bin\AntiCrack\3N.mhe 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다. (생성될 .hsb 파일과 동일한 위치)

✓ \Bin\HShield\HShield.dat 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다. (생성될 .hsb 파일과 동일한 위치)

➢ 참고: 3N.mhe 와 HShield.dat 파일을 복사하는 과정은 필수적인 것은 아니지만 핵쉴드 버전관리를 위하여 권장하고 있습니다.
(서버에 hsb가 존재하는 위치에 클라이언트에서 사용하는 3N.mhe 와 HShield.dat 파일을 올려둠으로써 3N.mhe 와 HShield.dat 버전이 올바르지 않은 클라이언트를 접속하지 못하게 관리할 수 있습니다)

✓ \Include\ AntiCpXSvr.h 헤더 파일이 필요합니다. 적당한 위치에 복사합니다.

❖ 클라이언트 관련 준비사항

✓ \Lib\HShield.lib 라이브러리 파일이 필요합니다. 적당한 위치에 복사한 다음 게임 클라이언트 프로젝트에 추가합니다.

✓ \Include\HShield.h 헤더 파일이 필요합니다. 적당한 위치에 복사합니다.

5 서버 적용

❖ 서버 프로젝트에 앞서 복사한 AntiCpXSvr.h 파일을 위치에 맞게 인클루드시킵니다.

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

❖ AHNHS_TRANS_BUFFER의 이해

서버에서 사용하는 _AhnHS_MakeRequest 함수와 클라이언트에서 사용하는 _AhnHS_MakeResponse 함수에서는 AHNHS_TRANS_BUFFER 구조체를 버퍼로 입력 받아 결과를 출력합니다. 버퍼의 구조는 아래와 같습니다. 실제 생성된 메시지는 byBuffer 배열에 저장되며 해당하는 배열의 길이는 nLength에 출력됩니다.

```
typedef struct _AHNHS_TRANS_BUFFER
{
    unsigned short nLength;
    unsigned char byBuffer[ANTICPX_TRANS_BUFFER_MAX/* 송수신 패킷의 최대 크기 */];
} AHNHS_TRANS_BUFFER, *PAHNHS_TRANS_BUFFER;
```

❖ 서버를 실행하기 전에 AntiCpXSvr 초기화를 실행합니다. (_AhnHS_CreateServerObject)

```
// ① 게임 서버가 초기화 되는 부분에 _AhnHS_CreateServerObject를 호출합니다.
void GameServer_OnInitialize()
{
    ....
    AHNHS_SERVER_HANDLE hServer = NULL;

    hServer = _AhnHS_CreateServerObject ( g_szHsbFilePath ); // .hsb 파일 경로

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

① g_szHsbFilePath는 HSBGen.exe를 통해 생성한. hsb파일의 전체 경로입니다.

❖ 클라이언트가 세션을 생성할 때마다 클라이언트 핸들을 생성합니다. (_AhnHS_CreateClientObject)

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

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

```
    // 해당 서버의 핸들을 이용해 접속한 유저의 핸들을 생성
    hClient = _AhnHS_CreateClientObject ( hServer );

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

❖ 클라이언트와의 통신을 처리하는 스레드 함수 내에 요구 메시지를 생성하는 처리 모듈을 추가합니다. (_AhnHS_MakeRequest)

```
    // 해당 유저의 핸들을 이용해 요청 메세지 작성
    // 각 유저의 요청 메세지는 해당 유저의 핸들을 이용해서 작성
    ulRet = _AhnHS_MakeRequest ( hClient, &stRequestBuf );

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

    // Socket을 통해 stRequestBuf 전송

    ....

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

❖ 클라이언트로부터 요청 메시지에 대한 응답 메시지가 전송되었을 때 이를 검증하는 모듈을 추가합니다. (_AhnHS_VerifyResponse)

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // 클라이언트로부터 수신한 응답 버퍼

    ....

    // Socket을 통해 stResponseBuf 수신

    ....

    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_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() = %X\n", ulRet );
    }

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

    ....
}
```

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

② 예제의 if 문에서 처리한 메시지가 아니라면 정상적인 경우이므로 계속 진행하고 그렇지 않다면 에러 처리한 후 해당 클라이언트의 소켓을 종료하고 접속 리스트에서 제거합니다.

❖ 클라이언트로부터 요청 메시지에 대한 응답 메시지가 전송되었을 때 이를 검증하는 함수인 _AhnHS_VerifyResponse 함수대신에 하기에 언급된 _AhnHS_VerifyResponseEx 함수를 이용할 수 있습니다.

```
void GameServer_VerifyReponse()
{
    AHNHS_TRANS_BUFFER stResponseBuf; // 클라이언트로부터 수신한 응답 버퍼

    ....

    // Socket을 통해 stResponseBuf 수신

    ....

    ulRet = _AhnHS_VerifyResponseEx ( hClient,
                                      stResponseBuf.byBuffer,
                                      stResponseBuf.nLength,
                                      *pnErrorCode );

    if ( ulRet == ANTICPX_RECOMMEND_CLOSE_SESSION )
    {
        //게임 클라이언트에서 해킹 행위가 감지되었으므로,
        //게임 클라이언트와의 접속을 차단하도록 합니다.
    }
```

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

    ....
}
```

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

② 함수의 반환값이 ANTICPX_RECOMMEND_KEEP_SESSION이면 정상적인 경우이므로 계속 진행하고 그렇지 않고 ANTICPX_RECOMMENT_CLOSE_SESSION이면 에러 처리한 후 해당 클라이언트의 소켓을 종료하고 접속 리스트에서 제거합니다.

❖ 클라이언트가 접속을 중지하여 세션이 종료되면 _AhnHS_CloseClientHandle 함수를 통해 클라이언트 핸들을 닫습니다. (_AhnHS_CloseClientHandle)

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

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

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

6 클라이언트 적용

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

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

    ....

    // Socket을 통해 stRequestBuf 수신

    ....

    ulRet = _AhnHS_MakeResponse ( stRequestBuf.byBuffer, stRequestBuf.nLength,
```

```
&stResponseBuf );

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

    // Socket를 통해 stResponseBuf 서버로 전송

    ....

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

- ✓ 서버로부터 요청 메시지를 수신하면 그에 해당하는 응답 메시지를 작성합니다.
- ✓ 클라이언트의 메시지 처리 루틴에 요청 메시지 수신 부분을 추가하고 _AhnHS_MakeResponse 함수를 호출하여 처리합니다.

7 적용 시 주의사항

- ❖ 디버깅 및 고객지원을 위하여 에러 메시지를 출력할 때는 에러 코드를 함께 출력하는 것이 좋습니다.
- ❖ 에러 발생 시 단순히 메시지만 발생시키면 그 상태로 게임을 계속 진행하는 경우가 있으므로 에러가 발생하면 반드시 클라이언트와의 접속을 종료하고 해당 클라이언트의 소켓 종료와 접속 리스트에서 제거하는 작업을 수행하십시오.
- ❖ 서버의 요청에 대한 응답이 없을 경우 커넥션을 종료하도록 구성해야 합니다. (권장 응답 시간: 10초)
- ❖ HSBGen.exe로 .hsb 파일을 생성할 때 클라이언트 실행 파일은 패킹되기 이전의 원본 파일이어야 합니다.
- ❖ Linux/Unix 환경에서는 대소문자를 구분하므로 HSPub.key 파일명은 정확하게 입력되어야 합니다.

8 테스트 및 배포

- ❖ 게임 클라이언트를 실행하고 간단한 테스트를 통해 확장 서버 연동 동작 여부를 판단할 수 있습니다.
  - ✓ Case 1. 클라이언트의 실행 파일을 에디터로 열어서 특정 부분을 변조한 다음 실행시켜서 서버에 접속했을 때 연결이 끊기는 것을 확인합니다.
- ❖ 배포 시에는 다음 과정에 따라 배포를 진행하는 것을 권장합니다.
  - ✓ 적용 및 테스트 ⇒ Quality Assurance 진행(AhnLab) ⇒ QA Report를 전달받은 후 수정사항 확인 후 최종 배포 버전 작성 ⇒ 배포