AhnLab

HackShield Pro

for Online Game

Quick Guide

V 2.0

서버 연동 기능

안철수연구소

1 시작하기 전에

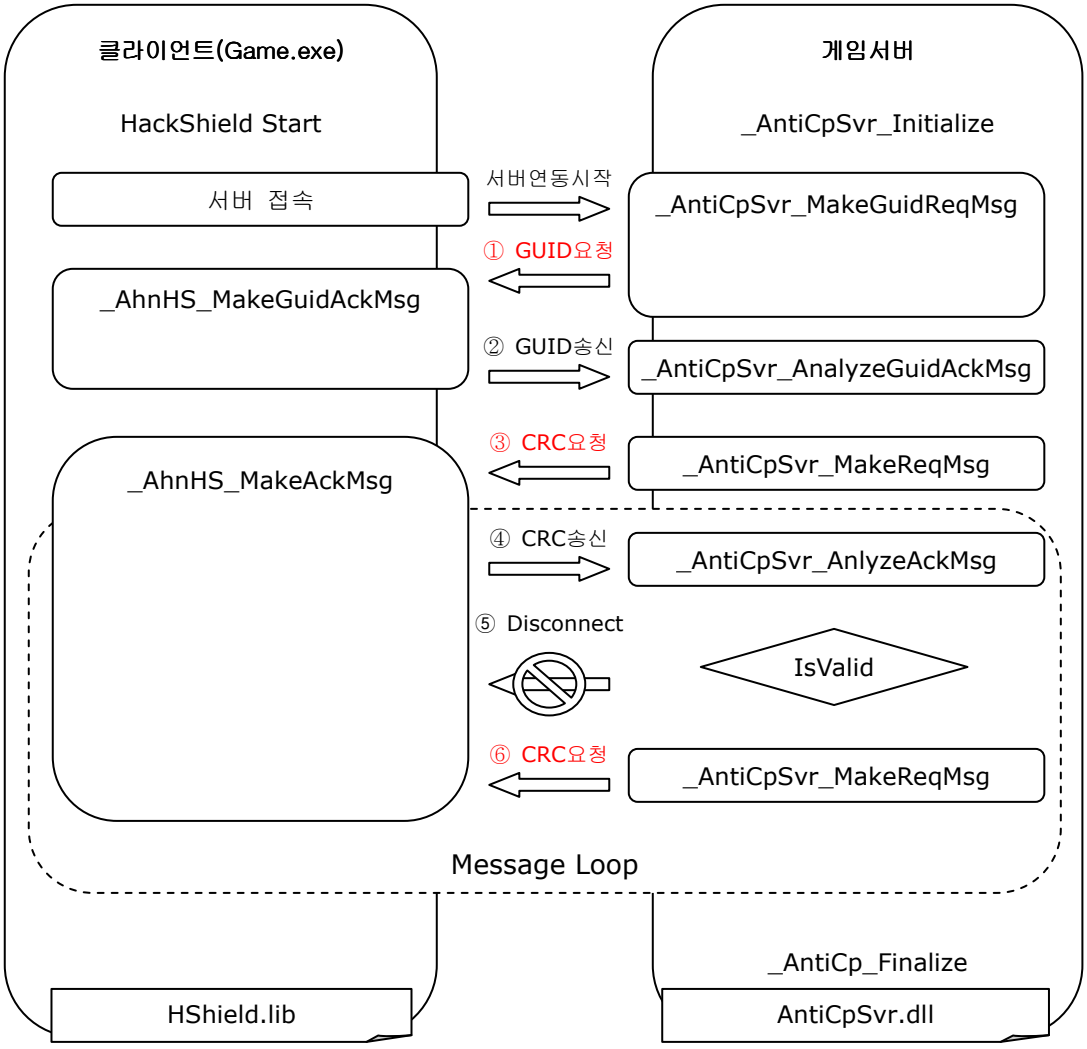
- 이 문서는 HackShield Pro(이하 핵쉴드) SDK 적용을 위한 가이드라인입니다. Microsoft Visual C/C++ 언어로 작성하는 것을 기준으로 작성되었으며, 핵쉴드의 서버 연동 기능에 대한 부분을 다룹니다.
- 처음 핵쉴드 코드 적용을 돕기 위하여 대략적인 설명과 간략한 샘플 코드를 다룹니다. 세부 구현은 각 게임 개발사의 정책에 맞추어 진행하십시오.
- \Doc 폴더 안에 *"AhnLab HackShield Pro 프로그래밍 가이드"* 문서가 제공됩니다. 이 문서는 HackShield Pro의 전체 구조와 기능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

2 HackShield Pro 서버 연동 기능의 구성 파일

- \Bin\AntiCrack\AntiCpSvr.dll : 서버 연동 dll 파일(Windows 용)
- \Bin\AntiCrack\AntiCpSvrTool.exe : CRC 생성 툴
- \Bin\AntiCrack\HSPub.key : 서버 연동 인증 키 파일
- \Bin\HShield\3N.mhe : 휴리스틱 엔진 버전 관리 파일
- \Bin\HShield\HShield.dat : 핵쉴드 버전 관리 파일
- \Include\AntiCpSvrFunc.h : 서버 연동 헤더 파일
- \Lib\AntiCpSvr.lib : 서버 연동 라이브러리 파일(Windows 용)
- \Bin\AntiCrack\Linux\libanticpsvr.so : 서버 연동 라이브러리 파일(Linux 용)

3 동작 구조

- 핵쉴드 서버 연동 통신 구조는 기존의 게임에 구현된 통신 구조를 그대로 사용합니다 (별도의 통 신모듈을 구현할 필요가 없습니다).
- 클라이언트의 버전을 체크하여 부정 실행을 방지합니다.
  - ✓ 서버는 연결된 클라이언트의 실행 파일 버전을 알기 위해 GUID 요구 메시지를 보내고 그에 대한 응답 메시지를 받습니다.
  - ✓ 핵쉴드가 제공하는 _AntiCpSvr_AnalyzeGuidAckMsg API를 통하여 버전 정보를 체크합니다.
  - ✓ 이때 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 일정 시간 간격으로 CRC 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 클라이언트가 서버로 접속하게 되면 서버는 클라이언트의 버전 정보를 확인하기 위하여 GUID 요구 메시지를 보냅니다.

② 클라이언트는 GUID 요구 메시지를 받아 이에 대응하는 버전 응답 메시지를 생성하여 서버로 보냅니다. 서버는 이 메시지를 분석하여 허용하는 버전이 아니라면 연결을 끊습니다.

③ CRC 요구 메시지를 클라이언트로 보냅니다.

④ 클라이언트에서 CRC 요구 메시지에 대응하는 응답 메시지(Ack)를 보내오면 서버는 이를 분석하여 클라이언트의 정상 여부를 판단합니다.

⑤ 이 과정에서 유효하지 않은 응답을 보내거나 응답을 보내지 않으면 서버는 클라이언트가 위, 변조되었다고 판단하여 연결을 끊습니다.

⑥ 유효한 응답을 보내면 클라이언트는 정상 실행 되며 서버는 일정 주기를 가지고 CRC 요구 메시지를 보내 게임 도중에 클라이언트를 변조하는 행위를 방지합니다.(④ ~ ⑥ 동작 반복)

4 적용 전 준비사항

- Windows 환경 서버 관련 준비사항
  - ✓ \Lib\AntiCpSvr.lib 라이브러리 파일이 필요합니다. 적당한 위치에 복사한 다음 게임 서버 프로젝트에 추가합니다.
  - ✓ \Bin\AntiCrack\AntiCpSvr.dll 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다.
  - ✓ \Bin\AntiCrack\HSPub.key 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다. (생성될 .crc 파일과 동일한 위치)
  - ✓ \Bin\AntiCrack\3N.mhe 파일이 필요합니다. 게임 서버의 실행 경로에 복사합니다. (생성될 .hsb 파일과 동일한 위치)
  - ✓ \Bin\AntiCrack\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\AntiCpSvrFunc.h 헤더 파일이 필요합니다. 적당한 위치에 복사합니다.
- Linux 환경 서버 관련 준비사항
  - ✓ \Bin\AntiCrack\Linux\libanticpsvr.so 라이브러리 파일이 필요합니다. 적당한 위치에 복사합니다.
  - ✓ \Include\AntiCpSvrFunc.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

- [주의] Linux 환경에서 작업할 때는 다음을 추가하거나 코드에서 직접 대체하여 사용하십시오. (서버가 Windows 환경일 경우에는 해당 사항 없음)

```
#define ERROR_SUCCESS 0

typedef unsigned char BYTE;
```

- 서버 프로젝트에 앞서 복사한 AntiCpSvrFunc.h 파일을 위치에 맞게 인클루드 시킵니다.

```
#include "AntiCpSvrFunc.h"
```

- 게임 서버를 시작할 때 서버 연동 초기화 함수(AntiCpSvr_Initialize)를 호출합니다.

```
// 게임서버 시작시 핵쉴드 서버모듈을 초기화합니다.
void GameServer_OnInitialize()
```

```
{
    ...

    // ① Client 정보가 저장된 HackShield.crc파일을 로드합니다.
    dwRet = _AntiCpSvr_Initialize ( szHashFilePath );

    if ( dwRet != ERROR_SUCCESS )
    {
        // 에러 처리
        // 권장사항 : 에러코드 로그 저장. 게임서버 종료
    }

    ...
}
```

① szHashFilePath는 AntiCpSvrTool을 통해 생성한 CRC파일의 전체 경로입니다.

❖ 클라이언트와의 통신을 처리하는 스레드 함수 내에 GUID 메시지를 처리하는 모듈을 추가합니다. (_AntiCpSvr_MakeGuidReqMsg, _AntiCpSvr_AnalyzeGuidAckMsg)

```
// 서버에 새로운 클라이언트가 접속하면 GUID를 확인한다.
void GameServer_OnLogOn()
{
    ....

    // SIZEOF_GUIDACKMSG는 GUID Ack Message를 담을 버퍼의 크기로
    // AntiCpSvrFunc.h에 정의되어 있습니다.
    BYTE        byGUIDAckMsg[SIZEOF_GUIDACKMSG] = { 0, };
    // SIZEOF_GUIDREQMSG는 GUID Request Message를 담을 버퍼의 크기로
    // AntiCpSvrFunc.h에 정의되어 있습니다.
    BYTE        byGuidReqMsg[SIZEOF_GUIDREQMSG] = { 0, };
    // SIZEOF_GUIDREQINFO는 GUID Request Info를 담을 버퍼의 크기로
    // AntiCpSvrFunc.h에 정의되어 있습니다.
    BYTE        byGUIDClientInfo[SIZEOF_GUIDREQINFO] = { 0, };
    // 주의: CrcInfo는 클라이언트가 접속되는 동안
    //       계속 유지되어야 하는 값입니다.
    HSHIELD_CLIENT_CONTEXT  CrcInfo = { 0, };

    // ① GUID 요청 메시지를 만듭니다.
    dwRet = _AntiCpSvr_MakeGuidReqMsg ( byGuidReqMsg, byGUIDClientInfo );

    // ② GUID 요청 메시지를 클라이언트로 전송합니다.
    SendToClient( byGuidReqMsg, SIZEOF_GUIDREQMSG );

    // ③ 클라이언트가 보낸 응답메시지를 받습니다.
    dwRet = ReceiveFromClient ( byGUIDAckMsg );

    if ( dwRet != ERROR_SUCCESS )
```

```
    {
        // 중요!! 에러처리
        // 만약 클라이언트가 응답이 없는 경우라면 비정상적인 클라이언트이므로
        // 접속을 차단합니다.
        DisconnectClient();
    }

    // ④ 클라이언트가 보내온 GUID AckMsg를 검증합니다.
    dwRet = _AntiCpSvr_AnalyzeGuidAckMsg ( byGUIDAckMsg ,
                                           byGUIDClientInfo,
                                           &CrcInfo );

    // ⑤ _AntiCpSvr_AnalyzeGuidAckMsg 함수의 리턴 값을 검사합니다.
    if ( dwRet != ERROR_SUCCESS )
    {
        // 중요!! 에러처리
        // 에러코드가 비정상적인 경우에는 클라이언트의 접속을 차단하는 것이 좋습니다.
    }

    ....
}
```

① 클라이언트가 연결되면 _AtiCpSvr_MakeGuidReqMsg 함수를 호출하여 클라이언트로 전달할 암호화된 버전 정보 요구 메시지를 생성합니다.

② 생성된 버전 요구 메시지를 클라이언트로 전송합니다.

③ 유효한 클라이언트라면 버전 요구 메시지를 받고 암호화된 버전 응답 메시지를 전송하게 됩니다.

④ 서버는 _AntiCpSvr_AnalyzeGuidAckMsg 함수를 호출하여 클라이언트로부터 받은 버전 응답 메시지를 분석합니다.

⑤ 버전이 정상적이라면(ERROR_SUCCESS) 계속 진행하고 그렇지 않다면 에러 처리한 후 해당 클라이언트의 소켓을 종료하고 접속 리스트에서 제거합니다.

❖ 클라이언트와의 통신을 처리하는 스레드 함수 내에 CRC 요구 메시지를 생성하는 처리 모듈을 추가합니다. (_AntiCpSvr_MakeReqMsg, _AntiCpSvr_AnalyzeAckMsg)

```
// ① 주기적으로 접속된 클라이언트에 대해서 무결성을 검증합니다.
void GameServer_OnCheckClient()
{
    ....
    static bool          bFirstCheck     = true;
    unsigned long     ulCheckOption = 0;

    // ulCheckOption 은 ReqMsg 의 마지막 인자로서 클라이언트에 요청하는 검사 옵션입니다.
    // 처음에는 ANTICPSVR_CHECK_ALL 을 설정하여 Memory, HShield File , Game File
    //, Engine File 을 검사하고 이후에는 ANTICPSVR_CHECK_GAME_MEMORY 만 설정 하
    // 여 Memory 검사만 주기적으로 합니다.
```

```
if  ( bFirstCheck == true )
{
      ulCheckOption = ANTICPSVR_CHECK_ALL;
      bFirstCheck = false;
}
else
{
      ulCheckOption = ANTICPSVR_CHECK_GAME_MEMORY;
}

// SIZEOF_ACKMSG는 Ack Message를 담을 버퍼의 크기로
// AntiCpSvrFunc.h에 정의되어 있습니다.
BYTE    byAckMsg[SIZEOF_ACKMSG] = { 0, };
// SIZEOF_REQMSG는 Request Message를 담을 버퍼의 크기로
// AntiCpSvrFunc.h에 정의되어 있습니다.
BYTE    byReqMsg[SIZEOF_REQMSG] = { 0, };
// SIZEOF_REQINFO는  Client Info 를 담을 버퍼의 크기로
// AntiCpSvrFunc.h에 정의되어 있습니다.
BYTE    byClientInfo[SIZEOF_REQINFO] = { 0, };

// ② 요청 메시지 만들기
// 주의:  CrcInfo는 GUID 요청과정에서 구해진 값으로
//        클라이언트가 접속되는 동안 계속 유지되고 있는 값입니다.
dwRet = _AntiCpSvr_MakeReqMsg ( &CrcInfo,
                                byReqMsg,
                                byClientInfo,
                                ulCheckOption );

// ③ 요청 메시지를 클라이언트로 전송합니다.
SendToClient( byReqMsg, SIZEOF_REQMSG );

// ④ 클라이언트부터 응답메시지를 받습니다.
dwRet = ReceiveFromClient ( byAckMsg );

if ( dwRet != ERROR_SUCCESS )
{
    // 에러처리
    // 만약 클라이언트가 응답이 없는 경우라면 접속을 차단합니다.
    DisconnectClient();
}

// ⑤ 클라이언트가 보내온 AckMsg를 검증한다.
dwRet = _AntiCpSvr_AnalyzeAckMsg ( &CrcInfo, byAckMsg , byClientInfo );

// ⑥ _AntiCpSvr_AnalyzeAckMsg 함수의 리턴 값을 검사합니다.
if ( dwError != ERROR_SUCCESS )
{
```

```
        DisconnectClient();
    }

    ....
}
```

① 정책에 맞게 일정한 주기를 결정하여 주기마다 검사하는 모듈을 호출하도록 합니다. 주기적 검사 과정을 통해 게임 클라이언트를 실행하는 중간에 메모리를 변조한다 해도 이를 감지할 수 있습니다. 주기가 짧을수록 더 빠르게 변조된 클라이언트를 감지할 수 있습니다 (MMORPG의 경우 2분 내외가 적당합니다).

② _AntiCpSvr_AnalyzeGuidAckMsg 함수를 호출하여 GUID 메시지에 대한 처리가 성공하면 _AntiCpSvr_MakeReqMsg 함수를 호출하여 클라이언트로 전달할 암호화된 CRC 요구 메시지(ReqMsg)를 생성합니다.

※ 중요: ReqMsg 생성시 함수의 마지막 인자로 MEMORY, FILE, HSHIELD에 대한 검사를 구분할 수 있습니다. 안전한 보호를 위하여 처음 한번만 ANTICPSVR_CHECK_ALL 옵션을 사용하고, 그 후부터는 게임 성능에 지장을 주지 않도록 ANTICPSVR_CHECK_GAME_MEMORY 옵션을 사용하는 것을 권장합니다.

③ 위에서 생성한 암호화된 요구 메시지를 클라이언트로 전송합니다.

④ 클라이언트에서 요구 메시지를 받고 이에 대한 응답 메시지를 서버로 보냅니다.

⑤ 서버는 _AntiCpSvr_AnalyzeAckMsg 함수를 호출하여 클라이언트로부터 받은 CRC 응답 메시지를 분석합니다.

⑥ 정상적인 응답 메시지라면(ERROR_SUCCESS) 계속 진행하고 그렇지 않다면 에러 처리한 후 해당 클라이언트의 소켓을 종료하고 접속 리스트에서 제거합니다.

❖ 게임 서버를 종료할 때 서버 연동 기능을 해제합니다. (_AntiCpSvr_Finalize)

```
// 게임서버 종료시 핵쉴드 서버모듈을 해제한다.
// 사용했던 자원과 메모리를 해제한다.
void GameServer_OnFinalize()
{
    ...

    _AntiCpSvr_Finalize ();

    ...
}
```

6 클라이언트 적용

❖ 보호함수 설정 (_AhnHS_SaveFuncAddress)

✓ 엔트리 포인트 함수 내에서 핵쉴드 서비스를 시작한 후 보호 함수를 설정합니다.

```
int nResult = _AhnHS_SaveFuncAddress( 3,
                                      HS_CallbackProc,
                                      HS_Init,
```

```
                                                    NetworkThreadProc );
```

- ✓ 첫 인자는 보호할 함수의 개수이며 이후의 인자들은 가변 인자로써 보호할 함수의 포인터를 입력합니다. 보호할 함수는 최대 32개까지 가능합니다.
- ✓ 보호할 함수는 실행 파일 내에 존재하는 함수여야 합니다.
  ※ 주의: DLL 내의 함수나 오버로딩 함수는 지원하지 않습니다.
- ✓ 반드시 핵쉴드 서비스를 시작(StartService)한 후에 호출되어야 합니다.
- ✓ 핵쉴드 함수 중에서는 Callback 함수와 Initialize 함수만 추가하면 됩니다.
- ✓ 그 외의 함수는 게임 클라이언트에 따라 중요한 역할을 하거나 해킹의 표적이 될만한 함수들을 추가하십시오.

❖ 서버와의 통신을 처리하는 스레드 함수 내에 GUID 요구 메시지에 대한 Ack 메시지 처리 모듈을 추가합니다. (_AhnHS_MakeGuidAckMsg)

```
// GUID Ack Msg를 작성합니다.
// 이 Msg는 서버에서 사용하는 _AntiCpSvr_MakeGuidReqMsg 에 대한 Ack Msg입니다.
BYTE byReqMsg(SIZEOF_GUIDREQMSG]= { 0, };
BYTE byAckMsg[SIZEOF_GUIDACKMSG] = { 0, };

dwRet = ReceiveFromServer( byReqMsg );

// GUID Ack Msg를 작성합니다.
dwRet = _AhnHS_MakeGuidAckMsg ( byReqMsg, byAckMsg );
if ( dwRet != ERROR_SUCCESS )
{
    // 에러 처리
    // 해당 함수가 성공이 아니라면 클라이언트를 종료시킵니다.
    ExitClient();
}
```

- ✓ 서버에서 GUID 요구 메시지가 전송되면 그에 해당하는 메시지를 작성합니다.
- ✓ 클라이언트의 메시지 처리 루틴에 GUID 요구 메시지 부분을 추가하고 _AhnHS_MakeGuidAckMsg 함수를 호출하여 처리합니다.

❖ 서버와의 통신을 처리하는 스레드 함수 내에 CRC 요구 메시지에 대한 Ack 메시지 처리 모듈을 추가합니다. (_AhnHS_MakeAckMsg)

```
// Ack Msg를 작성합니다.
// 이 Msg는 서버에서 사용하는 _AntiCpSvr_MakeReqMsg 에 대한 Ack Msg입니다.
BYTE byReqMsg[SIZEOF_REQMSG] = { 0, };
BYTE byAckMsg[SIZEOF_ACKMSG] = { 0, };

dwRet = ReceiveFromServer( byReqMsg );

// Ack Msg를 작성합니다.
dwRet = _AhnHS_MakeAckMsg ( byReqMsg, byAckMsg );
if ( dwRet != ERROR_SUCCESS )
{
```

```
    // 에러 처리
    // 해당 함수가 성공이 아니라면 클라이언트를 종료시킵니다.
    ExitClient();
}
```

- ✓ 서버에서 CRC 요구 메시지가 전송되면 그에 해당하는 메시지를 작성합니다.
- ✓ 클라이언트의 메시지 처리 루틴에 CRC 요구 메시지 부분을 추가하고 _AhnHS_MakeAckMsg 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.
- ❖ Linux/Unix 환경에서는 대소문자를 구분하므로 HSPub.key 파일명은 정확하게 입력되어야 합니다.

8 테스트 및 배포

- ❖ 적용 완료되었으면 서버와 클라이언트를 새로 빌드합니다.
- ❖ AntiCpSvrTool.exe를 이용하여 CRC파일을 생성합니다.
  - ✓ 패커를 사용한다면 반드시 실행 파일을 패킹한 후에 CRC 파일을 생성해야 합니다.
  - ✓ AntiCpSvrTool.exe 의 사용법은 *"AhnLab HackShield 프로그래밍 가이드: 8.4. AntiCpSvr 툴 사용 방법"*을 참고하시기 바랍니다.
  - ✓ CRC 파일 생성이 완료되면 서버에서 로딩할 수 있는 특정 경로에 복사합니다.
- ❖ 게임 서버를 실행합니다.
- ❖ 게임 클라이언트를 실행하고 간단한 테스트를 통해 서버 연동 동작 여부를 판단할 수 있습니다.
  - ✓ Case 1. 클라이언트의 실행 파일을 에디터로 열어서 특정 부분을 변조한 다음 실행시켜 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
  - ✓ 테스트 도중 생성된 hshield.log 파일은 삭제하고 배포하는 것이 좋습니다.