AhnLab HackShield

Quick Guide

Version 2.0

AhnLab HackShield for Online Game

Basic Features

AhnLab

1 Preface

- This document is a guideline for applying HackShield 2.0 (or HackShield) SDK. Please note that this document assumes that the application is written in C/C++ in Microsoft Visual 6.0 environment. It covers the basic functions of HackShield.
- This document provides sample codes to help understand how it works. You can use it according to your company's policies.
- For reference, a sample code "MiniA" is provided under the Sample folder.
- *"AhnLab HackShield 2.0 Programming Guide"* can be found in the Doc folder. This document provides the detailed information on AhnLab HackShield 2.0 and API functions. For more information, refer to this document.

2 HackShield Pro SDK

- \Bin : HackShield executable files
- \Data : HackShield data files
- \Developer: Files for developers for applying HackShield
- \Doc : User guide
- \Include : Include files for applying HackShield
- \Lib : Library files for applying HackShield
- \PatchSet : Patchset files for update
- \Sample : Sample files for reference

3 Getting Started

- Add library files.
  - ✓ HackShield uses the basic Windows libraries – version.lib and winmm.lib. Add these two library files to the project.
  - ✓ \Lib\HShield.lib Library file is required. Copy the file and add to the project.
- \Include\HShield.h header file is required. Copy the file to a proper folder.
- A license key is required. It consists of four-digit game code and 24-digit string. A unique license key is issued for each game.

4 HackShield Service

- Include the HShield.h file as follows:

```
#include "HShield.h"
```

- Add the function to call \HackShield service to the entry-point function of the game.
  - ✓ HackShield code is applied in the following order: initialization → start → game execution → stop → uninitialization. Before initializing the game client, initialize and start HackShield module. When terminating the HackShield module, stop and uninitialize the module.

```
WinMain (…)
{
    if ( !HS_Init() )
    {                                                    Initialize
        HS_UnInit();
        return 0;
    }

    if ( !HS_StartService() )
    {
        HS_StopService();                                Start Service
        HS_UnInit();
        return 0;
    }

    // Game execution routine
    ...

    HS_StopService();                                    Stop Service

    HS_UnInit();                                         Uninitialize
}
```

- ✓ If Uninitialize is not properly called, the code may not be properly executed. **Make sure Uninitialize(Stop) is called when Initialize(Start) fails and when there is an exception during game execution.**
- ✓ To protect the game client, it is recommended to immediately start HackShield service right after HackShield is initialized.

❖ Write HackShield initialization function (_AhnHS_Initialize).

```
BOOL HS_Init()
{
    int             nRet = HS_ERR_OK;
    TCHAR           szFullFilePath[MAX_PATH];
    TCHAR           szMsg[MAX_PATH];
    DWORD           dwOption = 0;

    // ① Specify EhSvc.dll path under HackShield folder.
    lstrcat ( szFullFilePath, _T( "\\HShield\\EhSvc.dll" ) );

    // ② Define option flag to be used to call _AhnHS_Initialize function
    dwOption = AHNHS_CHKOPT_ALL;

    // ③ Call _AhnHS_Initialize function and initialize HackShield service.
    nRet = _AhnHS_Initialize ( szFullFilePath,
                               HS_CallbackProc,          // callbackfunction
                               1000,                     // game code
                               "B228F291B7D7FAD361D7A4B7",   // license key
                               dwOption,                 // option flag
                               AHNHS_SPEEDHACK_SENSING_RATIO_NORMAL };

    // ④ Check the return value of _AhnHS_Initialize function.
    if ( nRet != HS_ERR_OK )
    {
```

```
            switch ( nRet )
            {
            case HS_ERR_COMPATIBILITY_MODE_RUNNING:
            case HS_ERR_NEED_ADMIN_RIGHTS:
            case HS_ERR_INVALID_FILES:
            case HS_ERR_INIT_DRV_FAILED:
            case HS_ERR_DEBUGGER_DETECT:
            case HS_ERR_NOT_INITIALIZED:
            default:
                wsprintf( szMsg, "An error occurred in the hack prevention feature.(%x)", nRet );
                break;
            }
            MessageBox( NULL, szMsg, szTitle, MB_OK );
            return FALSE;
        }
        return TRUE;
}
```

① Set the absolute path for EhSvc.dll file of HackShield SDK in szFullFilePath parameter. This file is usually located under HShield folder. It is recommended to use GetModuleFileName function to properly set the absolute path. Do not use GetCurrentDirectory function.

② The flag defined here is used as a fifth factor when calling _AhnHS_Initialize function. Set the basic functions of HackShield. In order to debug the game, delete a few options before execution. For more information about each flag, refer to *AhnLab HackShield 2.0 Programming Guide: 2.4. Application Programming Interface*. It is recommended to use the flags above for testing and distribution, unless there are specific issues.

③ To call _AhnHS_Initialize function, it requires a game code and a license key. The last factor sets the speed hack detection sensitivity. Speed hack detection sensitivity depends on the game developer's policies. In general, AHNHS_SPEEDHACK_SENSING_RATIO_ NORMAL is used. For more information on each factor, refer to *"AhnLab HackShield 2.0 Programming Guide: 2.4. Application Programming Interface*.

④ Check the return value of _AhHS_Initialize function. If the execution is not normal (HS_ERR_OK), return an error message according to error code. For more information on each error message, refer to *"AhnLab HackShield 2.0 Programming Guide"*.

ex) In the case of HS_ERR_INVALID_FILES:

```
wsprintf (szMsg, "You have installed the wrong file.\nPlease reinstall the program. (%x)",
nRet );
break;
```

❖ Write HackShield service starting function (_AhnHS_StartService).

```
BOOL HS_StartService()
{
    int             nRet = HS_ERR_OK;
    TCHAR           szMsg[MAX_PATH];

    // ① Start HackShield service by calling _AhnHS_StartService function.
    nRet = _AhnHS_StartService();

    // ② Check the return value of _AhnHS_StartService function.
```

```
    if ( nRet != HS_ERR_OK )
    {
        switch ( nRet )
        {
        case HS_ERR_START_ENGINE_FAILED:
        case HS_ERR_DRV_FILE_CREATE_FAILED:
        case HS_ERR_REG_DRV_FILE_FAILED:
        case HS_ERR_START_DRV_FAILED:
        default:
            wsprintf ( szMsg, "An error occurred in the anti-hacking feature.(%x)", nRet );
            break;
        }
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        return FALSE;
    }
    return TRUE;
}
```

① In order to start HackShield service, AhnHS_Initialize function must be called. Anti-hacking feature starts when _AhnHS_StartService function is called. It is recommended to start HackShield service before the game initialization routine is executed.

② Check the return value of _AhHS_StartService function. If the result is not normal (HS_ERR_OK), return an error message according to error code and terminate the program. For more information on each error message, refer to *"AhnLab HackShield Programming Guide"*.

❖ Write HackShield service stopping function. (_AhnHS_StopService)

```
BOOL HS_StopService()
{
    Int   nRet = HS_ERR_OK;

    // ① Stop HackShield service by calling _AhnHS_StopService function.
    nRet = _AhnHS_StopService();

    if ( nRet != HS_ERR_OK )
    {
        return FALSE;
    }
    return TRUE;
}
```

① In order to stop the HackShield service, the HackShield service must be running. Call _AhnHS_StopService function. Then the HackShield service will be terminated, and the anti-hacking and hacking tool detection features are also disabled.

❖ Write the HackShield service uninitializing function. (_AhnHS_Uninitialize)

```
BOOL HS_UnInit()
{
    int   nRet = HS_ERR_OK;

    // ① Stop HackShield service by calling _AhnHS_Uninitialize function.
    nRet = _AhnHS_Uninitialize();
```

```
    if ( nRet != HS_ERR_OK )
    {
        return FALSE;
    }
    return TRUE;
}
```

① If the game client program is abnormally terminated, the HackShield anti-hacking driver may not be unloaded. Stop the service using Uninitialize. When terminating the client or making exceptions for normal/abnormal termination after calling the callback function, Uninitialize shall be executed.

② For abnormal uninitialization, add the following code at the beginning of the game in order to execute the HackShield stopping routing when the game is terminated.

```
::SetUnhandledExceptionFilter ( Game_UnhandledExceptionHandler );
```

③ Call HackShield StopService and Uninitialize in Game_UnhandledExceptionHandler() function registered above. This is only for when there is an exception.

❖ Write event transmission function for anti-hacking and hacking tool detection features. (_AhnHS_Callback)

```
int __stdcall HS_CallbackProc ( long lCode, long lParamSize, void* pParam )
{
    TCHAR          szMsg[MAX_PATH];
    // ① Display a proper error message for each case
    switch ( lCode )
    {
    // ② Engine Callback
    case AHNHS_ENGINE_DETECT_GAME_HACK:
        wsprintf( szMsg, “You cannot run the following program and game simultaneously. (%x) \n [%s]”,
                lCode, (LPTSTR)pParam );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ③ AutoMacro Detection
    case AHNHS_ACTAPC_DETECT_AUTOMACRO:
        wsprintf(szMsg, _T("Suspected macro operation has been detected. (Code = %x"),
                lCode);
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ④ No special processing
    case AHNHS_ACTAPC_DETECT_AUTOMOUSE:
    case AHNHS_ACTAPC_DETECT_ALREADYHOOKED:
        break;
    // ⑤ Speed
    case AHNHS_ACTAPC_DETECT_SPEEDHACK:
        wsprintf( szMsg, “Suspected speed hack operation has been detected. (%x)”, lCode );
        MessageBox( NULL, szMsg, szTitle, MB_OK );
        break;
    // ⑥ Debugging Prevention
    case AHNHS_ACTAPC_DETECT_KDTRACE:
    case AHNHS_ACTAPC_DETECT_KDTRACE_CHANGED:
```

```
            wsprintf( szMsg, “Attempt of game debugging has been detected. (%x)”, lCode );
            MessageBox( NULL, szMsg, szTitle, MB_OK );
            break;
        // ⑦ Other Abnormal Hacking Prevention features
        case AHNHS_ACTAPC_DETECT_DRIVERFAILED:
        case AHNHS_ACTAPC_DETECT_HOOKFUNCTION:
        case AHNHS_ACTAPC_DETECT_MODULE_CHANGE:
        case AHNHS_ACTAPC_DETECT_LMP_FAILED:
        case AHNHS_ACTAPC_DETECT_MEM_MODIFY_FROM_LMP:
        case AHNHS AHNHS_ACTAPC_DETECT_ENGINEFAILED:
        case AHNHS_ACTAPC_DETECT_CODEMISMATCH:
        case AHNHS_ACTAPC_DETECT_ANTIFREESERVER:
        case AHNHS_ACTAPC_DETECT_ABNORMAL_HACKSHIELD_STATUS:

            wsprintf( szMsg, “There is a problem in the hack detection feature. (%x)”, lCode );
            MessageBox( NULL, szMsg, szTitle, MB_OK );
            break;
        }
        return 1;
}
```

① Process the data detected by HackShield. Define each case in compliance with the game rules. For more information on each case, refer to *“AhnLab HackShield 2.0 Programming Guide: 2.4. Application Programming Interface”*.

② This is when a hacking program or potentially harmful program is detected. Notify the name of the detected program to the user. Show a message and terminate the game client.

③ This is when HackShield has detected hacking tools related to AutoMacro. Terminate the game client.

④ It is recommended not to show a message box or error message.. If necessary, record logs. In case of AHNHS_ACTAPC_DETECT_ALREADYHOOKED, some APIs are already hooked, and hooking may be made in some normal programs including security programs. In case of AHNHS_ACTAPC_DETECT_AUTOMOUSE, callback will not be called because it is a protection function, not a detection function.

⑤ This is when a speed hack is suspected. It is highly likely that it is a speed hack. Show a message, and judge whether to terminate the game client depending on the game developer’s policies.

⑥ This is when there is debugger trace. It is highly likely that debugging is currently being made for the game program. Show a message and terminate the game client.

⑦ This is when an error occurred in the message hooking or module change feature. Show a message and terminate the game client.

5 Caution

- Initialization options can be selectively used depending on the game developer’s policies. It is recommended to use the options used in the example in order to keep the game client safe.
- It is recommended to display an error messages with error codes for debugging and customer support.
- MessageBox(NULL, …) is used for messages. However, when running a game in full screen mode, the message box may get hidden behind the game. It is recommended to show the message in the

game screen. If you want to use MessageBox function, use Handle instead of Null.

- ❖ Users may ignore the message and continue playing the game. Show a message and forcibly terminate the game client.

6 Test and Distribution

- ❖ Compile the project and create a game client.
- ❖ Create HShield folder under the game client folder.
  - ✓ ex) …\[Game Directory]\HShield
- ❖ Copy files under \Bin\HShield folder to HShield folder.
- ❖ Verify whether HackShield functions normally by running the game client and conducting some tests:
  - ✓ Case 1. Execute Process Explorer and check whether dll information is displayed.
  - ✓ Case 2. Rename EhSvc.dll file in HShield, and check whether it is detected.
  - ✓ Case 3. Start a hacking tool and check whether it is detected.
- ❖ If no error is found, pack executable files.
  - ✓ Prevent easy debugging of the executable file using a packer.
  - ✓ HackShield provides a basic upx packer.
  - ✓ ex) upx.exe –f Source.exe –o Output.exe
- ❖ It is recommended to apply server-side detection as well. (Refer to Quick Guide for Server-side Detection.)
- ❖ Conduct initial distribution as follows:
  - ✓ Build and test ⇒ Quality assurance (by AhnLab) ⇒ Receive QA report, review, and write final version for distribution ⇒ Distribute
  - ✓ Delete the hshield.log file created during a test run before distribution.