1 Using Ad-Aware's Public API

Ad-Aware Public API Information for Developers

Note! This information is intended for developers who wish to develop and share plug-ins for Ad-Aware.

In a similar spirit to the Community Translations Project, we are giving developers the chance to create their very own plug-ins for Ad-Aware. To upload plug-ins, sign-up to the MyLavasoft community at www.lavasoft.com, go to the plug-ins page, and join in the fun!

Details for Developers

- Ad-Aware can only have one scan in progress at a time. If the scan method is called while Ad-Aware is busy, the scan is ignored. Applications will have to implement some sort of queue mechanism themselves to handle this issue.
- In order to clean an infected file, a GUI scan must be performed. This is the same as manually scanning the file using Ad-Aware.
- Applications will need to locate "aawapi.dll" and load it. This file can be found in the "... \Lavasoft\Ad-Aware\" folder. This file should not be moved. Currently, this path cannot be found in the Windows registry (due to various reasons).
- Avoid disconnecting if a "scan" or "GUI scan" operation is being performed.
- All relative paths will be converted to absolute paths with environment path of the running application as its base. See GetFullPathName as a reference.
- AAW-API is thread safe, so every thread (and process) can connect and disconnect independently of each other without any interference, except that only one scan can be performed at any one time.

API
Flags (adaware_flag_e)

- ADAWARE_NO_FLAGS = 0,
- ADAWARE_AUTO_START = 1,

  Tries to start the service if unable to connect

Description: Use OR (|) to combine flags

Results (adaware_res_e)

- ADAWARE_SUCCESS = 0
  - The request was successful
- ADAWARE_UNABLE_TO_CONNECT
  - For some reason, aawapi can not establish a connection to Ad-Aware
- ADAWARE_NO_SUCH_FILE
  - Trying to scan a none existing file or directory
- ADAWARE_SCAN_FAILED
  - The scan did not successfully complete, discard the returned result
- ADAWARE_NOT_CONNECTED
  - Trying to use a method without a succeeding call to adaware_connect

- ADAWARE_ALREADY_CONNECTED
  - Trying to establish a new connection without closing the first (see adaware_disconnect)
- ADAWARE_BUSY
  - Ad-Aware is currently busy and can't scan your requested file
- ADAWARE_LAUNCH_GUI_FAILED
  - The gui couldn't be started
- ADAWARE_NOT_FOUND
  - Ad-Aware doesn't seem to be installed, can happen in the aawapi.dll isn't in the Ad-Aware folder or that an old version of Ad-Aware is installed.

Timeouts (adaware_timeout_e)

- ADAWARE_CONNECT_SINGLE_TRY_TIMEOUT = 0,
- ADAWARE_CONNECT_DEFAULT_TIMEOUT = 30000

Scan result

```
typedef struct {
wchar_t path[1024];                    //path of infected file
wchar_t category[256];                 //category of infection
wchar_t family_name[256];              //name of the infection family
wchar_t family_description[2048];      //description of the infection family
adaware_infection_type infection_type;  //type of infection
int TAI;                               //threat assessment index: 0 < TAI <= 10 if infection found
} adaware_scan_results_t ;
```

Functions

adaware_res_e adaware_connect(adaware_flag_e, DWORD timeoutMs)

Description: Starts a connection to the Ad-Aware service, has to be called before any other adaware functions. It should never be called when a connection is already in place. timeoutMs is the connection timeout in milliseconds, a value smaller than or equal to 0 defaults to ADAWARE_CONNECT_DEFAULT_TIMEOUT.

Parameters: ADAWARE_NO_FLAGS, ADAWARE_AUTO_START

Return codes: ADAWARE_SUCCESS, ADAWARE_UNABLE_TO_CONNECT, ADAWARE_NOT_FOUND, ADAWARE_ALREADY_CONNECTED

Example:

```
adaware_res_e res = adaware_connect(ADAWARE_NO_FLAGS, 20000); //20 second timeout
```

or

```
adaware_res_e res = adaware_connect(ADAWARE_AUTO_START, 1000); //1 second timeout, start ad-aware if needed
```

adaware_res_e adaware_disconnect(adaware_flag_e)

Description: Closes the connection to the adaware service.

Parameters: ADAWARE_NO_FLAGS

Return codes: ADAWARE_SUCCESS, ADAWARE_NOT_CONNECTED

Example:

```
adaware_res_e res = adaware_discconnect(ADAWARE_NO_FLAGS);
```

adaware_res_e adaware_scan_file(const wchar_t *filepath, adaware_scan_result_t &result, int &malicious)

Description: Scans a given file and populate result with useful data.

Note: Malicious is either 0 or 1, but we are using an int instead of a bool to provide C compatibility. If adaware_scan_file doesn't return ADAWARE_SUCCESS, the data in result will be undefined.

Return codes: ADAWARE_SUCCESS, ADAWARE_NOT_CONNECTED, ADAWARE_NO_SUCH_FILE, ADAWARE_SCAN_FAILED,

Example:

```
adaware_scan_result_t scanres;

int malicious;

adaware_res_e res = adaware_scan_file(L"pathtofile", &scanres, ADAWARE_NO_FLAGS, &malicious);

if(res == ADAWARE_SUCCESS) {
 //do something with the scanres data
}
```

adaware_res_e adaware_scan_file_with_gui(const wchar_t *filepath)

Description: Scans a given file or directory with the adware gui. The caller will not access the scan results. This is currently the only way to let Ad-Aware handle (remove) any infections.

Return codes: ADAWARE_SUCCESS, ADAWARE_NOT_CONNECTED, ADAWARE_BUSY, ADAWARE_LAUNCH_GUI_FAILED

Example:

```
adaware_res_e res = adaware_scan_file_with_gui(L"pathtofile");
```

adaware_res_e adaware_is_busy()

Description: Check if any scan is currently in progress

Return codes: ADAWARE_SUCCESS, ADAWARE_NOT_CONNECTED, ADAWARE_BUSY

Example:

```
adaware_res_e res = adaware_scan_file_with_gui(L"pathtofile");
if(res == ADAWARE_BUSY) {
  // then we should "wait" until next scan
}
```

void adaware_set_mock(void *config) (not public)

Description: Replaces the service connection with a mock object with the given config, only for testing purposes.

Example:

```
ServiceClientMockConfig *mock = new ServiceClientMockConfig();
mock->canConnect = false;

mock->canAutoStart = true;
adaware_set_mock(mock);
//run test
```

1.1 API Samples

Two API sample applications: File scanner and CMD Test Application.

File scanner sample

Description: A simple test application demonstrating the basic functionality of the Lavasoft Ad-Aware API.
Files: file_scanner_sample.cpp
Usage: <appname>.exe AAWPath file_to_scan
AAWPath: The Ad-Aware install directory.
file_to_scan: Path to the file to be scanned.

Code:

```
/*
//      file_scanner_sample.cpp
//
//      Description:    A simple test application demonstrating the basic functionality
//                                  of the Lavasoft Ad-Aware API
//
//      Remarks:            Make sure that aawapi.dll is located in your Ad-Aware directory
//
//      Usage:              <appname>.exe AAWPath file_to_scan
//
//      Authors:            Anders Olofsson and Simon Edwardsson
*/
#include <windows.h>
#include <iostream>
/* include the Ad-Aware api for all the exported functionality */
#include "Ad-AwareAPI.h"
/*
Functor typedefs for Ad-Aware,
see the documentation for all exported functions
```

```
*/

typedef adaware_res_e (WINAPI* adaware_connect_cbp)(adaware_flag_e, signed int);

typedef adaware_res_e (WINAPI* adaware_disconnect_cbp)(adaware_flag_e);

typedef adaware_res_e (WINAPI* adaware_scan_file_cbp)(const wchar_t*,
adaware_scan_results_t *, int *);

/* Making the functor pointers global */

adaware_connect_cbp _adaware_connect;

adaware_disconnect_cbp _adaware_disconnect;

adaware_scan_file_cbp _adaware_scan_file;

/* Holds the API dll */

HINSTANCE hDLL;

/* Result messages (for debugging) */

wchar_t *resCodes[] =  {L"ADAWARE_SUCCESS",

        L"ADAWARE_UNABLE_TO_CONNECT",

        L"ADAWARE_NO_SUCH_FILE",

        L"ADAWARE_SCAN_FAILED",

        L"ADAWARE_NOT_CONNECTED",

        L"ADAWARE_ALREADY_CONNECTED",

        L"ADAWARE_BUSY",

        L"ADAWARE_LAUNCH_GUI_FAILED",

        L"ADAWARE_NOT_FOUND"};

/*  Initializes the functor pointers

Returns 0 upon success */

int Init(std::wstring &path)
{
        std::wstring dll_path = path;
        dll_path += L"\\";
        dll_path += L"aawapi.dll";

        /* Load API dll */
        hDLL = LoadLibraryW(dll_path.c_str());

        /* If Load was successful */
        if (hDLL != NULL)
        {
```

```
		/* Get function adresses: */

		_adaware_connect = (adaware_connect_cbp)GetProcAddress(hDLL,
							"adaware_connect");

		_adaware_disconnect = (adaware_disconnect_cbp)GetProcAddress(hDLL,
							"adaware_disconnect");

		_adaware_scan_file = (adaware_scan_file_cbp)GetProcAddress(hDLL,
							"adaware_scan_file");

		/* If anything failed */
		if (!_adaware_connect || !_adaware_disconnect || !_adaware_scan_file)
		{
			std::wcout<<L"FATAL: error occured while loading library function" <<
std::endl;

			FreeLibrary(hDLL);
			return 1;
		}
	}
	else /* Failed loading dll */
	{

		std::wcout << "FATAL: could not load awwapi.dll, sorry" << std::endl;
		return 1;
	}

	return 0;
}

void usage()
{
	std::cout << "usage: example_app.exe path_to_adaware file_to_scan" << std::endl;
}

int wmain(int argc, wchar_t* argv[])
{

	if(argc < 3)
	{
		usage();
		return EXIT_FAILURE;
	}

	std::wstring aaw_path	= argv[1];
	std::wstring file_path	= argv[2];

	/* Initialization */
	int initRes = Init(aaw_path);

	if(initRes != 0)
		return EXIT_FAILURE;

/*

	Connect and print result
```

```
		If Ad-Aware is not started, it will autostart.

	*/

		int connectRes = _adaware_connect(ADAWARE_AUTO_START,
	ADAWARE_CONNECT_DEFAULT_TIMEOUT);
		std::wcout<<L"adaware_connect: "<<resCodes[connectRes]<<std::endl;

		/* Scan and print result */

		adaware_scan_results_t scanResult;

		int infection;

		int scanRes = _adaware_scan_file(file_path.c_str(), &scanResult, &infection);
		std::wcout<<L"adaware_scan_file: "<<resCodes[scanRes]<<std::endl;

		/* If infection, print type */

		if(infection != 0)
			std::wcout<<L"Infection found! Family name: "<<scanResult.family_name<<std::
	endl;
		else
			std::wcout<<L"No infections found"<<std::endl;

		wprintf(L"infection: %d\n", infection);

		/* Disconnect and print result */

		int disconnectRes = _adaware_disconnect(ADAWARE_NO_FLAGS);
		std::wcout<<L"adaware_disconnect: "<<resCodes[disconnectRes]<<std::endl;

		/* Free dll */

		FreeLibrary(hDLL);
		return EXIT_SUCCESS;

	}
```

CMD Test Application

Description: An extended sample application capable of scanning files and folders, using both GUI scan and normal scan.
Files: CMDTestApplication.cpp
Usage: <appname>.exe AAWPath action [files]
AAWPath: The Ad-Aware install directory.
action:
/scan [filepaths] scan files and print results
/scandir [folderpath] scans the directory recursively
/scangui [path] scans the file or folder with the AAW GUI
Example: CMDTestApplication.exe "C:\Program Files\Lavasoft\Ad-Aware\ /scan "C:\file.txt"

Code:

```
/*
//    CMDTestApplication.cpp
//
//    Description:    An extended test application demonstrating all the functionality
//                    of the Lavasoft Ad-Aware API
//
//    Remarks:        Make sure that aawapi.dll is located in your Ad-Aware directory
//
//    Usage:          <appname>.exe AAWPath action [files]
//
//                    AAWPath:    The Ad-Aware install directory.
//                    action:
//                            /scan [filepaths]       scan files and print results
//                            /scandir [folderpath]   scans the directory recursively
//                            /scangui [path]         scans the file or folder with the AAW GUI
//
//                    Example: CMDTestApplication.exe "C:\Program Files\Lavasoft\Ad-Aware\ /scan "C:
\file.txt"
//
//    Authors:        Anders Olofsson and Simon Edwardsson
*/

#include "Ad-AwareAPI.h"
#include <iostream>
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

/* first of we create some callbacks for our dll functions */
typedef adaware_res_e (WINAPI* adaware_connect_cbp)(adaware_flag_e, signed int);
typedef adaware_res_e (WINAPI* adaware_disconnect_cbp)(adaware_flag_e);
typedef adaware_res_e (WINAPI* adaware_scan_file_cbp)(const wchar_t*,
adaware_scan_results_t *, int *);
typedef adaware_res_e (WINAPI* adaware_scan_file_with_gui_cbp)(const wchar_t*);

/* if you want to call adaware_connect you will now write _adaware_connect(); */
```

```
adaware_connect_cbp _adaware_connect;
adaware_disconnect_cbp _adaware_disconnect;
adaware_scan_file_cbp _adaware_scan_file;
adaware_scan_file_with_gui_cbp _adaware_scan_file_with_gui;

/* this is our function for scaning a file, dealing with the result and possible errors */
int scan_action(wchar_t *file) {
   wprintf(L"Begin scanning \"%s\"...\n",file);

   /* scan the file */
   adaware_scan_results_t scan_result;
   int found;
   int res = _adaware_scan_file(file, &scan_result, &found);

   /* if scan did not succeed:*/
   if(ADAWARE_SUCCESS != res) {
      switch(res) {
         case ADAWARE_NO_SUCH_FILE:
            wprintf(L"Can not locate (maybe a directory?): \"%s\"\n", file);
            break;
         default:
            wprintf(L"Scanning failed, error code: %i\n", res);
            break;
      }
   } else { /*scan succeded */
      /* is there anything to report? please note that scan_result is undefined if found is false */
      if(found) {
         wprintf(L"Path: %s\nInfection type: %i\n", scan_result.path, scan_result.infection_type);
         wprintf(L"Familyname: %s\nThreatlevel: %i\n", scan_result.family_name, scan_result.TAI);
         wprintf(L"Category: %s\nFaimlyDescription: %s\n", scan_result.category, scan_result.
family_description);
      } else {
         wprintf(L"\"%s\" seems to be clean\n", file);
      }

   }

   return 0;
}

/* scans a directory recursively. Call with depth = 0*/
void scan_dir(const wchar_t *filepath, int depth) {
      /* don't scan more than 20 directories recursively*/
      if(depth > 20)
         return;

      WIN32_FIND_DATA findFileData;
      HANDLE hFind;

      /* scan the provided directory */
      wchar_t *tmp = new wchar_t[wcslen(filepath) + wcslen(L"/* ")];
      wsprintf(tmp, L"%s\\*", filepath);
      hFind = FindFirstFile(tmp, &findFileData);
      delete[] tmp;

      if(hFind  == INVALID_HANDLE_VALUE) {
         wprintf(L"No file(s) found\n");
```

```
            return;
        }

        if(depth == 0) {
            /* init the adaware connection */
            int res = _adaware_connect(ADAWARE_NO_FLAGS, -1);
            if(ADAWARE_SUCCESS != res) {
                printf("FATAL: Could not connect to adaware (%i)\n", res);
                return;
            }
        }

        do {
            /* we do not care about . and .. since they are no real files */
            if(!wcscmp(findFileData.cFileName,L".") || !wcscmp(findFileData.cFileName, L".."))
                continue;

            /* if file is a directory: */
            if(findFileData.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY) {
                wchar_t *tmp = new wchar_t[wcslen(filepath) + wcslen(L"\\ ") + wcslen(findFileData.
cFileName)];
                wsprintf(tmp, L"%s\\%s", filepath, findFileData.cFileName);
                /* recursive scan */
                scan_dir(tmp, depth + 1);
                continue;
            }

            /* not a directory, scan this file */
            wchar_t *tmp = new wchar_t[wcslen(filepath) + wcslen(L"\\ ") + wcslen(findFileData.
cFileName)];
            wsprintf(tmp, L"%s\\%s", filepath,findFileData.cFileName);
            scan_action(tmp);
            delete[] tmp;
            wprintf(L"+--------------------------------+\n");

            /* loop as long as there are files left*/
        } while(FindNextFile(hFind, &findFileData));

        if(depth == 0) {
            /* destroy the connection */
            _adaware_disconnect(ADAWARE_NO_FLAGS);
        }
}

/* scans the specified file or directory using the Ad-Aware GUI*/
int scan_with_gui(const wchar_t *path)
{
    /* init the adaware connection */
    int res = _adaware_connect(ADAWARE_NO_FLAGS, -1);
    if(ADAWARE_SUCCESS != res) {
        printf("FATAL: Could not connect to adaware (%i)\n", res);
        return 0;
    }

    /* scan path with gui*/
    res = _adaware_scan_file_with_gui(path);
```

```
    if(res == ADAWARE_BUSY)
        std::wcout << L"Ad-aware is currently busy, please try again later" << std::endl;
    else if(res != ADAWARE_SUCCESS)
        std::wcout << L"Scanning failed" << std::endl;

    /* destroy the connection */
    _adaware_disconnect(ADAWARE_NO_FLAGS);
    return 0;
};

/* prints how the application should be used */
int usage(wchar_t *app) {
    wprintf(L"Usage:\t%s AAWPath action [files]\n\n"
            L"\tAAWPath:\n"
            L"\t\t[Ad-Aware install directory]\n"
            L"\taction:\n"
            L"\t\t/scan [filepath]\tscan file and print results\n"
            L"\t\t/scandir [filepath]\tscan all files in filepath directory and print results\n"
            L"\t\t/scangui [filepath]\tscan a file using the Ad-Aware GUI",
            app, app);
    return 0;
}

/* main function */
int wmain(int argc, wchar_t* argv[])
{
    /* parse and handle input: */
    wchar_t *action = 0;
    wchar_t *options[10];
    int nr_options = 0;
    for(int i = 1; i < argc; i++)
    {
        if(argv[i][0] == '/')
            action = argv[i];
        else
        {
            /* no more than 10 arguments to a command! */
            if(i > 2 && nr_options < 10)
                options[nr_options++] = argv[i];
        }
    }

    /* we have to specify an action */
    if(!action) {
        return usage(argv[0]);
    }
    /* Load the dll */
    std::wstring aawpath = argv[1];
    aawpath.append(L"\\aawapi.dll");
    HINSTANCE hDLL = LoadLibraryW(aawpath.c_str());
    wprintf(L"Path to dll: \t%s\n", aawpath.c_str());

    if (hDLL != NULL)
    {
        /* then we load the functions */
        _adaware_connect = (adaware_connect_cbp)GetProcAddress(hDLL,
                                        "adaware_connect");
```

```
        _adaware_disconnect = (adaware_disconnect_cbp)GetProcAddress(hDLL,
                                    "adaware_disconnect");
        _adaware_scan_file = (adaware_scan_file_cbp)GetProcAddress(hDLL,
                                    "adaware_scan_file");
        _adaware_scan_file_with_gui = (adaware_scan_file_with_gui_cbp)GetProcAddress(hDLL,
                                    "adaware_scan_file_with_gui");
        /* If anything failed */
        if (!_adaware_connect || !_adaware_disconnect || !_adaware_scan_file || !
_adaware_scan_file_with_gui)
        {
            std::wcout << "FATAL: error occured while loading library function" << std::endl;
            std::cout << (int)(_adaware_connect) << (int)(_adaware_disconnect) << (int)
_adaware_scan_file << std::endl;
            FreeLibrary(hDLL);
            return 1;
        }
    } else {
        std::wcout << "FATAL: could not load awwapi.dll, sorry" << std::endl;
        return 1;
    }

    /* ACTION: SCAN DIRECTORY */
    if(!wcscmp(action, L"/scandir"))
    {

        if(nr_options != 1)
        {
            wprintf(L"scandir needs exactly one argument");
            return 1;
        }
        //wprintf(L"options: %d\n", nr_options);
        scan_dir(options[0], 0);
    }
    /* ACTION: SCAN WITH GUI */
    else if(!wcscmp(action, L"/scangui"))
    {
        if(nr_options != 1)
        {
            wprintf(L"/scangui needs exactly one argument\n");
            return 1;
        }
        /* add path to a temp string*/
        std::wstring tmp = L"";
        tmp += options[0];
        /* scan file with gui */
        scan_with_gui(tmp.c_str());

    }
    /* ACTION: SCAN FILE(-S) */
    else if(!wcscmp(action, L"/scan"))
    {
        if(nr_options < 1)
        {
            wprintf(L"/scan needs at least one argument\n");
            return 1;
        }
```

```
        /* init the adaware connection (before the scans, rather than connecting once for each scan)
*/
        int res = _adaware_connect(ADAWARE_NO_FLAGS, 5000);
        if(ADAWARE_SUCCESS != res)
        {
            printf("FATAL: Could not connect to adaware (%i)\n",res);
            return 0;
        }
        /* make a scan for every file specified */
        for(int i = 0; i < nr_options; i++)
        {
            res = scan_action(options[i]);
        }
        /* destroy the connection */
        _adaware_disconnect(ADAWARE_NO_FLAGS);
        return 0;
    }
    else
    {
        /* default:  print usage*/
        return usage(argv[0]);
    }

    return EXIT_SUCCESS;
}
```