Universal anticheat just for your game!

Contents

HackTrap Simplified Implementation 2

1. Preparation 2
2. Adding code to the Visual Studio Project 2
3. Digitally signing required files 3

Updating HackTrap 4

Me or one of my users has problem! 4

HackTrap extras to-do 5

1. Closing in case you found any problem 5
2. Using HACKTRAP_INITIALIZED macro 5
3. Using HACKTRAP_CHECK_INTEGRITY macro 5
4. Using HACKTRAP_CHECK_INTEGRITY_EX macro 5

HackTrap ADVANCED Implementation 6

1. Initializing and Loading 6
   a. Initialization & Loading 6
   b. Heartbeat Callback (Swiss army knife callback) 6
2. File signing 7
3. Error messages (HTShowMessage.exe) 8
4. Advanced usage 8
5. Troubleshooting 9
6. GDPR-related information 9

HackTrap Simplified Implementation

1. Preparation

You should have received the following three things:

- HackTrap.zip or Publish.zip
- Digital_keys.zip
- Your license (text)

WARNING! NEVER GIVE OUT YOUR KEYS ANYWHERE OR ANYONE. IT'S CALLED PRIVATE FOR A GOOD REASON. YOUR SERVER MAY GET PERMANENTLY BLOCKED IN SUCH A CASE.

Steps:

a) You should extract HackTrap.zip somewhere. Then extract the digital_keys.zip into your previously extracted folder's Tools directory – next to the FileSigner.exe. You will use this tool with the extracted privkey.dat and pubkey.dat at a later stage.
b) Copy the whole HackTrap folder into your game client
c) Copy the Source/Client files (every cpp and h file) into your UserInterface source directory next to the other such files

2. Adding code to the Visual Studio Project

- Open the Visual Studio project
- Right click the UserInterface -> Add -> Existing item…
- Add the earlier copied .cpp and .h files to the project
- Open UserInterface.cpp
- Insert the following code after your usual includes:

```
#define HACKTRAP_LICENSE "ADD LICENSE HERE"
#include "HTDefaultImplementation.h"
```

ADVANCED SIDENOTES:

If you have **CEF (Chrome embedded framework)** in your client (by default you don't have, you have to manually code it or buy it from someone.) – then you should also add this extra define:

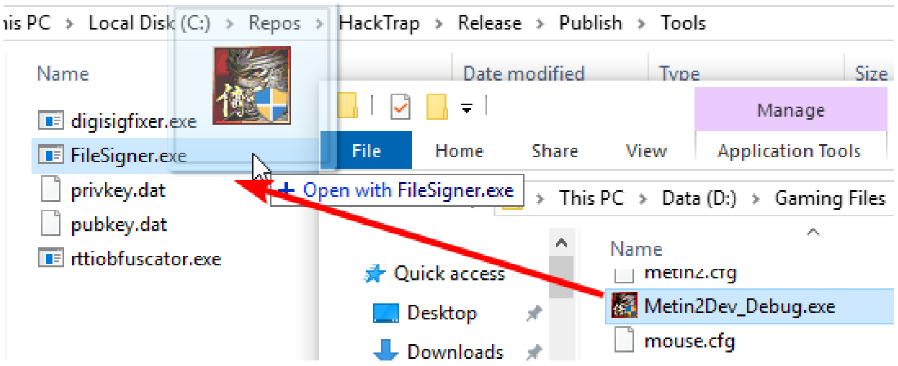
```
#define HACKTRAP_PRECALL [](){ CefWebBrowser_Startup(GetModuleHandleW(nullptr)); }()
```

HackTrap will call this line before initializing the anticheat. Now you have to find where else you call this "`CefWebBrowser_Startup(…)`" function and delete that. Only my code should initialize it only once.

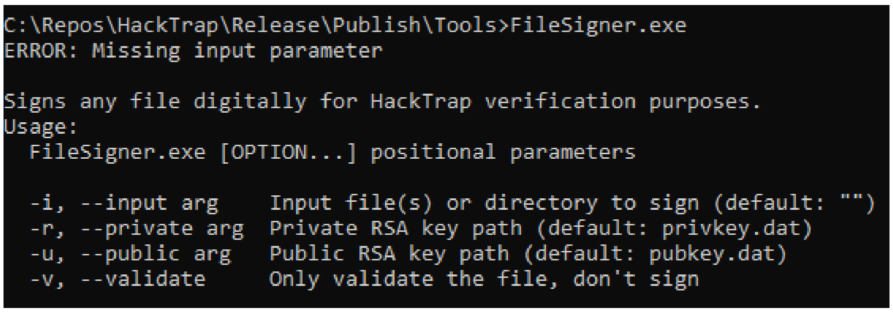
3. Digitally signing required files

You have to digitally sign a few files using my tool. This is pretty easy. **There are 3 ways to do this:**

1. Drag the file you want to sign into the FileSigner.exe like this:



2. Double Click the exe
3. Use command line:



```
C:\Repos\HackTrap\Release\Publish\Tools>FileSigner.exe
ERROR: Missing input parameter

Signs any file digitally for HackTrap verification purposes.
Usage:
  FileSigner.exe [OPTION...] positional parameters

  -i, --input arg    Input file(s) or directory to sign (default: "")
  -r, --private arg  Private RSA key path (default: privkey.dat)
  -u, --public arg   Public RSA key path (default: pubkey.dat)
  -v, --validate     Only validate the file, don't sign
```

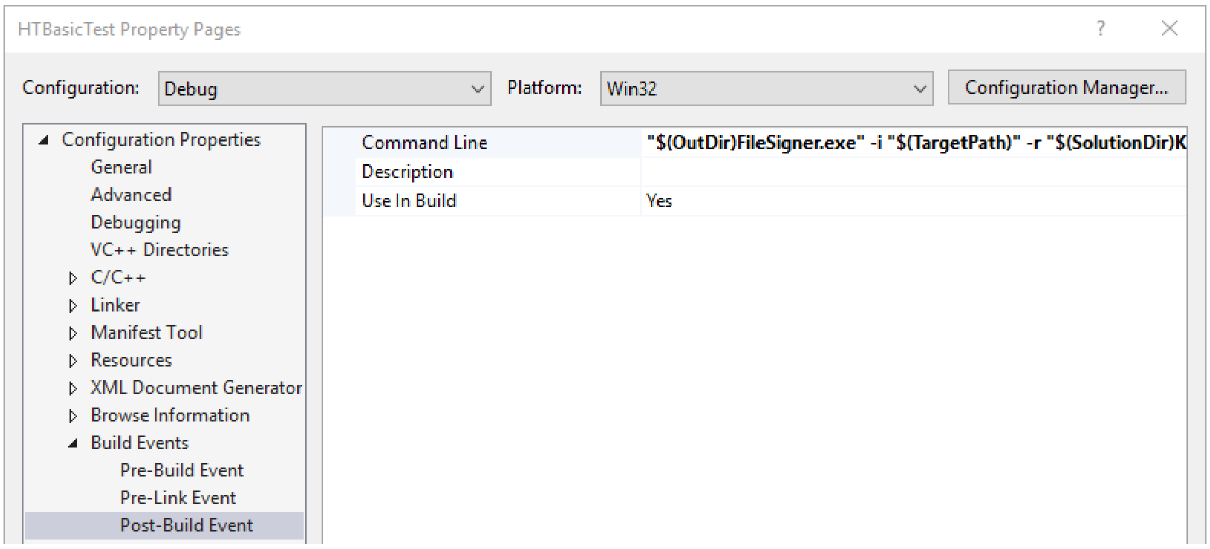
IMPORTANT!

List of the files you have to digitally sign:

- gameclient.exe (you can rename it anytime)
- every file from the '**miles**' directory or equivalent
- every '.pyd' file from the **'lib'** directory or equivalent
- *If you have patcher protection enabled, then your patcher exe (patcher, updater etc)*

ADVANCED SIDENOTES:

You can use the Visual Studio Post-Build event step combining the the FileSigner.exe's command line to automatize the signing of your executable like this:



Configuration: Debug   Platform: Win32

Command Line	"$(OutDir)FileSigner.exe" -i "$(TargetPath)" -r "$(SolutionDir)K
Description	
Use In Build	Yes

Updating HackTrap

If you receive an update, probably it will be a single zip file, which contains several directories. The update procedure is a really simple thing to do, because all you have to do is to:

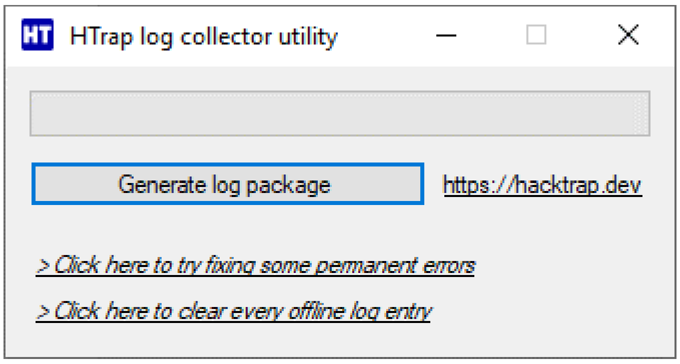
1) Copy-paste the HackTrap directory into your game folder
2) Be careful you didn't overwrite your translated ini file!
3) Copy-paste the Source files into your project
4) Rebuild your whole project

That's all, you're done.

Also, if I forgot to send invitation, please ask for an entry into the private discord group where I'm distributing all the updates.

Me or one of my users has problem!

Don't worry. HackTrap has pretty useful error messages and error logs! There's a log collector tool in the HackTrap directory, which looks like this:


HTrap log collector utility
Generate log package
https://hacktrap.dev
> Click here to try fixing some permanent errors
> Click here to clear every offline log entry

First please try to press the "Fixing some permanent errors" text, because a few cheats are modifying your system in a way that I can't distinguish the computer from a real cheater. This button will reset those.

If it doesn't help then screenshot the error message you get and also "Generate a log package" and send it to me. Then I'll be able to investigate where the issue is and how to exactly fix those.

Leaving these log files readable would give hints to hackers, that's why I'm writing encrypted logs which are only decryptable by me.

Also make sure you didn't forget to sign your exe and your license is not expired 😊 If you or your client has any problems feel free to contact me anytime on discord.

HackTrap extras to-do

1. Closing in case you found any problem

If you found any error or any macro returned that there's an error, I suggest not printing text into logs, because they are very easy to find and then they are going to be bypassed. I suggest calling:

```
HackTrap::BasicHeartbeat::Terminate(custom_error_number, HACKTRAP_DIRECTORY)
```

Where the custom error number is a number you randomly think of, and HackTrap directory is the directory where the anti-cheat is placed. You can skip the last parameter and just call it like:

```
HackTrap::BasicHeartbeat::Terminate(123456);
```

So, you know when 123456123 is the error message, then that specific event happened.

2. Using HACKTRAP_INITIALIZED macro

This macro can be used to detect if HackTrap is loaded successfully. I suggest checking for it a few times, but not necessarily in a loop. Example usage:

```
if (!HACKTRAP_INITIALIZED) {
    // Do something, for example call HackTrap::BasicHeartbeat::Terminate
}
```

3. Using HACKTRAP_CHECK_INTEGRITY macro

This macro can be used widely in your source. If you put this somewhere then a HackTrap integrity check will be run! If some hacker is able to suspend my anticheat from running, then this call will force a new integrity check to run, making their bypass useless.

There are multiple levels of this integrity check. You can use these versions:

```
HACKTRAP_CHECK_INTEGRITY(HACKTRAP_INTEGRITY_SLOW);
HACKTRAP_CHECK_INTEGRITY(HACKTRAP_INTEGRITY_FAST);
```

A fast version usually takes around 1-10ms per run, while the slow can even take like 300ms or more. Also there's no guarantee that it will not run for any longer on very slow machines, but the code tries to be as fast as possible.

4. Using HACKTRAP_CHECK_INTEGRITY_EX macro

This is a new version of the previous macro, it uses more stealthy methods to detect modifications to the anticheat or to detect if the anticheat is bypassed.

```
DWORD local = 0;
HACKTRAP_CHECK_INTEGRITY_EX(HACKTRAP_INTEGRITY_SLOW, &local, 1234567);
if (local != 1234567) { ... }
```

You can check the variable a bit later as well, so it's not that suspicious and harder to find!

HackTrap ADVANCED Implementation

These notes here are only good to know if you want to make some very custom implementation which differs from the usual. Normally you would not need it and you can feel free to stop reading from here.

1. Initializing and Loading

HackTrap loading is divided into three parts:

- Initializing your server’s settings
- Loading the anti-cheat
- Optional: Handling heartbeat messages

a. Initialization & Loading

Requires your license in a string variable and **optionally** you can specify your heartbeat callback here (please see below about the callback). Over the time, this callback is used for much more. Nothing special thing is required:

```
bool success = HackTrap::Initialize("ENTER LICENSE HERE");
```

Please always check the return value, which is a bool. It indicates if initialization succeeded. If it didn’t, then you should exit. In this case you can retrieve an error code, which is very useful for debugging purposes:

```
DWORD code = HackTrap::GetLastError();
```

Loading happens with the following very straightforward function:

```
bool success = HackTrap::Start();
```

By default the Start() function uses the usual “HackTrap.dll” name inside a folder called “HackTrap”, but you can feel free to rename it by adding a simple string parameter anytime. Also always check if it returns success!

b. Heartbeat Callback (Swiss army knife callback)

Originally it was intended only as a heartbeat, but recently it’s used for much more! As described earlier you should initialize the settings with an extra callback parameter:

```
bool success = HackTrap::Initialize("ENTER LICENSE HERE", &HeartbeatCallbackFunc);
```

Requires a STATIC, NON-MEMBER function as parameter, which handles heartbeat communication. Exact typedef can be seen in HTSDK.h and here:

```
typedef uint32_t(CALLBACK* HackTrapCallback)(
	IN const Command		command,
	IN const uint8_t* const		message		OPTIONAL,
	IN const uint32_t		messageSize	OPTIONAL
)
```

Example implementation can be found in the HTDefaultImplementation.h file.

2. File signing

Every file which is associated with HackTrap is signed with a digital signature. The following files are always come PRE-SIGNED, so you should NEVER touch them:

- HackTrap dll
- HackTrap.htdb

The following files are checked for YOUR digital signature, which is included in your license. Using this you can prevent modifications to these files:

- The Patcher/Launcher executable
- The Game executable itself
- Every file that is executable and loaded into the game, but not in the main directory of the game (not next to the executable) – For example: .mix files are practically renamed dll files. Same applies to .pyd files. They are renamed dlls as well.

After signing these files, you MUST not modify the files at byte level. You may rename it, but never change it in any ways.

These files are all checked at runtime for their signature. If any of the fails the integrity check, then the game terminates. This includes that the **PATCHER MUST BE RUNNING** for a few seconds after launching the game, because HackTrap validates its parent, which should be the launcher/patcher. If it's not your patcher, but a hack, or a no-patch launcher (which skips patching, only starts your game), then it will have NO digital signature and game will terminate.

To sign files, you'll need:

- Pubkey.dat
- Privkey.dat
- FileSigner.exe

FileSigner.exe comes with a command line help, you can get further information there.

WARNING! NEVER GIVE OUT YOUR KEYS ANYWHERE OR ANYONE. IT'S CALLED PRIVATE FOR A REASON.

3. Error messages (HTShowMessage.exe)

Yet there aren't too much of error messages, but it's under implementation to expand them. Error messages are shown by launching HTShowMessage.exe with parameters, which is related to the specific error.

You can either create your own software with own UI or use the default HTShowMessage. This latter, my tool is just a very simple tool which shows a MessageBox and reads the text which should be opened from the *HackTrap.Language.ini* file. If you're fine with it, then you won't need any further coding!

All you have to do is to edit the *HackTrap.Language.ini* file and translate the messages.

HackTrap code and dll is programmed to automatically call this error message tool when any issue happens, so you don't have to care about that!

If you want to create a new software on your own code, all you have to do is to parse the command line arguments. There are 2 parameters:

1) Error category (number)
2) Extended error code (string)

Neither may contain space or quote mark in it.

4. Advanced usage

a) HACKTRAP_FUNC_MARKER macros

Not required, but very strongly suggested to use something like this!

If you're using a protection that utilizes markers (such as VMProtect, The Engima Protector or any kind of Oreans product), then you can utilize the `”HACKTRAP_FUNC_MARKER_START”` and `“HACKTRAP_FUNC_MARKER_END”` macros because they're inserted at the critical points of the SDK. If you define them before including the SDK, it will automatically use them.

It's not necessary, but a good thing, to hide the loading of the Anticheat. I suggest using some kind of code virtualizing feature, which most of current executable protectors has (CodeVirtualizer, Themida, VMProtect, Enigma and some more).

b) RTTIObfuscator.exe

Most C++ executables have a so-called RTTI information embedded inside, which easily tells where your encryption and other C++ routines rely. It's possible to avoid using RTTI, but most external libraries, like CryptoPP and others are relying on it. The best is to hide these strings, so attackers won't be able to easily tag your code. Let them reverse it for a few weeks 😉

Usage is pretty self-explaining, just call it from command line, help is available.

c) digisigfixer.exe

You only have to use this special tool if you're buying a real Digital Code Signing Certificate for a few hundred bucks on the internet. (Digicert, Comodo, etc)

5. Troubleshooting

HackTrap is continuously logging in the background. Any error message related to HackTrap is saved to the user's machine in an encrypted form. These encrypted logs can be retrieved (along with some user machine information) using the HackTrapLogCollector.exe. If a user runs this, then he's able to generate an encrypted log package, which helps finding exactly what error the user experienced.

6. GDPR-related information

Please note that the Anticheat has a call-home mechanism, which means it's sending **anonymous** information to its server located at https://hacktrap.dev

It's mainly used as telemetry data to ensure the perfect working and detection of any cheat-related info. Nothing personal is transmitted to the server.

The log packages contain a few non-identifiable information in them, which includes some hardware & OS information. Also, it contains the CWD which may include some sensitive data, but very unlikely.