licensingSystem Documentation

sleepundeflow

v0.0.1

Abstract

This project is not to be used for malicious or illegal purposes. Project is available on a github repository `https://github.com/sleepunderflow/licensingSystem`

Table of contents

1 Overview **2**

2 Development rules **3**

3 Injected Values **4**
3.1 Definition 4
3.2 Fields 4
3.3 Flags 4

4 Embedded Tools **5**
4.1 Format 5
4.2 Main Header 5
4.3 Individual headers 6
4.4 Items 6

5 Configuration File **7**
5.1 Item 7
5.2 Command 8

6 Conditional Instructions **9**

7 Report **10**

8 FAQ **11**

Chapter 1

Overview

The goal of this project is to create a tool that allows for easy and secure method of giving someone a bunch of tools and commands to run, collecting results from it (potentially in the future wits some sort of scripting language aloowing for things like conditional commads) and giving a single encrypted file for the person being helped to transfer to you for review

Disclaimer: This project is not intended to be used for malicious purposes and is only meant to make helping others easier, more efficient and protecting your knowledge.

Chapter 2

Development rules

Suggestions:

- Server can be done in python3 flask framework (I can provide hosting after we leave dev phase) (needs more research into the framework itself)
- I'm still working on specs, If you have any idea let me know/add them to the docs
- Any suggestions? Say it

Suggested development rules:

- use Issues on GitHub
- keep docs up to date whenever you change something (if you don't want to play with LaTeX just leave a txt file describing changes, I can add it later)
- keep it nice and clean
- Make sure to periodically check documentations. Especially dev hints and specs

Chapter 3

Injected Values

3.1 Definition

There is a structure in the client binary defined as follows:

```
struct sInjectedConfig {
  uint64_t header              = 0xDEADBEEFDEADBEEF;
  uint64_t flags               = 0x0000000000000000;
  uint64_t injectedDataOffset  = 0x1111111111111111;
};
```

It is statically initialised with those arbitrary values as it is meant to be overwriten by an injector. The header will be detected and then the corresponding values injected.

3.2 Fields

header - A magic value to be detected by an injector. It has to be at the beginning of the structure.

flags - A bitmap for enabling / disabling specific features. Explanation in section 3.3

injectedDataOffset - The offset in file to the beginning of the Main Header for the Embedded Tools (chapter 4)

3.3 Flags

Currently defined bit flags:

- **bit 0** - `FLAG_EMBEDPRESENT` - if set, Embedded Tools are appended to the file
- **bit 1** - `FLAG_EMBEDENCRYPT` - if set, the encryption of Embedded Tools is enabled. Otherwise just extract
- **bit 2** - `FLAG_ELEVATE` - if set, try to elevate privileges

The rest is currently reserved for a future use.

Chapter 4

Embedded Tools

This chapter explains the format for embedded executables for the client.

4.1 Format

Embedded executables and scripts are appended to the end of a client binary by the injector script.

The format of that section is as follows:

- Main Header
- Item 1 Header
- Item 1
- Item 2 Header
- Item 2
- ...

4.2 Main Header

The main header of embedded tools is defined as follows:

```
struct embeddedToolsMainHeader {
  uint32_t totalSize;
  uint32_t numberOfItems;
  char contentHash[64];
};
```

Explanation:

totalSize - the full size of embedded tools in bytes (including main header and individual headers)

numberOfItems - the number of embedded Items

contentHash - the SHA256 hash of the embedded part (Individual headers + items)

4.3 Individual headers

The individual header is separate for each embedded item and placed directly before the item content starts.

It is defied as follows:

```
struct individualHeader {
  uint32_t ID;
  uint32_t length;
  uint32_t flags;
  char fileName[64];
  char itemHash[64];
  char permissions[4];
};
```

Explanation:

ID - numeric ID of the item. Starts from 0 and is being incremented by 1.

length - length of the item (excluding individual header)

flags - bitmap that contains settings per item. Currently used bits (bit order in file:|7-0||15-8| - 16-bit integer in little-endian format):

- 0: remove-after-use: 0-false, 1-true
- 1-31: reserved for future use

fileName - the file name of the item including extension

itemHash - SHA256 hash of the item after unpacking and decrypting

permissions - item permissions to be set by *chmod*, last byte shall be null byte

4.4 Items

The item itself is going to be encrypted using one of the individual keys generated and injected using the Injector script. It'll be decrypted when extracting. It'll only be extracted when will have to be used.

Chapter 5

Configuration File

The injector script is configured by a configuration file *config.conf* located in the same directory as the injector script itself.

Format for the file is as follows:

```
[Item]
configuration for item object

[Command]
command configuration

[Item]
...
```

There is no particular order for command/item entries. The only thing is that they will be added to the binary in the same order as entered to the config file but as they will be separated it doesn't matter whether you first add all the items then commands or command - item - command or however you like.

Configuration options are being added in format key="value"

if value should contain " character it has to be followed by \ (\")

5.1 Item

Item settings are in a following format:

```
[Item]
name="injected item name"
path="path to a file to inject"
permissions="permissions"
remove-after-use="true/false"
```

Path is a relative path to a file to be injected.

Permissions is a desired file permission in a chmod format (for example 755).

Name is the injected item name that can be used internally instead of the ID in the [Command] structure. Has to be a string and can't start with a digit.

If remove-after-use is set to true or missing, the extracted file will be deleted when the client terminates.

5.2 Command

Command settings are in the following format:

```
[Command]
command="command to be executed"
save-output="true/false"
print-output="true/false"
condition="condition"
loop="false/true"
```

Command is the desired shell command to execute. It can use the **{ID}** structure to use the injected items ([Item] part) If such structure is in the command, the required item will be extracted to a temporary location and {ID} will be replaced with the path to the extracted file. The ID is either an integer pointing to which [Item] in order is to be used starting from 0 or a string being the name of the required file ([Item]->name). If the name is used it will be replaced with the index while generating.

If print-output is false the output from the command will not be shown on the terminal. By default it's set to true.

If print-output is true (default) the report from running the command will be generated (chapter 7)

If condition is present, the command will only be run if the condition is true. The condition format is specified in chapter 6

If loop is true (implicit default is false) then the command will be repeated until condition becomes false

Chapter 6

Conditional Instructions

TO DO

Chapter 7

Report

TO DO

Chapter 8

FAQ

- **Will you add a function to run it quietly/fully automatically/send results automatically in the background?** No, this would make it much easier to use it for malicious purposes and that's not a goal of this project. This way at least a 2nd targeted party has to make some actions which makes for some form of a consent I guess.
- **What is the actual goal of this project?** No idea. Fun? Gaining knowledge?
- **Why aren't you using a library for that thing? Why don't you use X or Y instead of Z, it's easier?** The goal of this project is to learn how things work under the hood. You won't learn using just ready-made tools.