Quest Journals
Journal of Software Engineering and Simulation
Volume 3 ~ Issue 7 (2017) pp: 14-21
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Simple Obfuscation Tool for Software Protection

*Ján Hurtuk[1], Branislav Madoš[1]
[1]*Department Of Computers And Informatics, Faculty Of Electrical Engineering And Informatics Technical University Of Košice, Letná 9, 042 00 Košice, Slovak Republic,*
*Corresponding author: *Ján Hurtuk*

Received 10 August, 2017; **A**ccepted 12 August, 2017 © The author(s) 2017. **P**ublished with open access at **www.questjournals.org**

ABSTRACT: *This paper discusses the issue of source code obfuscation and also the creation of a tool for automatic obfuscation of source code written in C language. The result is a tool that performs both data flow and control flow obfuscation and allows the user to configure the applied transformation algorithm. For easier and better usability the tool provides a graphical user interface, which brings possibility to control and configure transformation process.*

Keywords: *C language, Obfuscator, Parser, Control flow obfuscation, Data flow obfuscation*

I. INTRODUCTION

Malicious code is one of the indispensable components of today's world where digitization is almost in every industry, so it is important that antivirus software manufacturers do not wait for the attackers, but make progress by developing newer, better and more sophisticated security mechanisms by creating new types of malicious software. Since a variety of antivirus programs use the method of fingerprints or signatures for quick scan of individual files, the use of automated obfuscators, can be considered a powerful tool in searching for various modifications of specific malicious code samples.

II. OBFUSCATION CATEGORIES

Figure 1 shows several kinds of intellectual property protections contained in the source code. One of them is also obfuscation. Obfuscation is a process that involves editing the source code to disable static and dynamic code analysis while maintaining the original functionality in the transformed code. Source code analysis is strongly connected with process of reverse engineering, which uses disassemblers and decompilers to reveal data structures and control flow of source code.

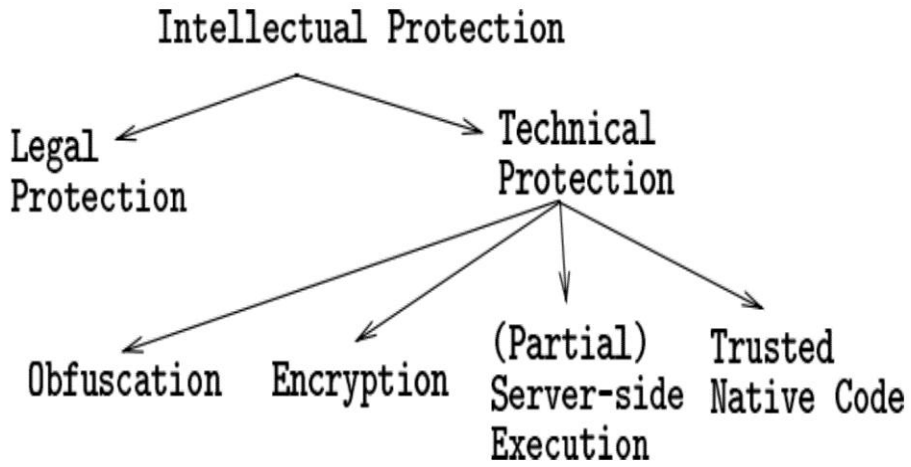

Figure 1 Intellectual Protection Categories.

According to the basic taxonomy from document [1] shown in Figure 2, obfuscation transformations are divided into four categories depending on the target information.

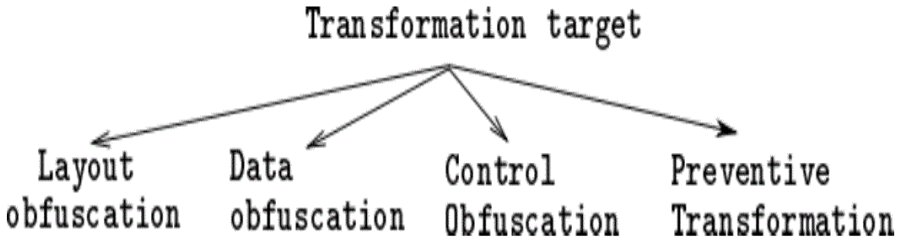

Figure 2 Obfuscation Categories.

A. Layout Obfuscation: This type of obfuscation aims on modifying those information in code, which are not necessary for the execution or does not affect the code functionality when removed. This includes trivial transformations focused on making the code harder to be understood by human reader, for example: renaming identifiers, removing comments and formatting or removing statement used for logging purposes [2].

B. Data flow obfuscation: Aims on modifying data structures in the given code. It contains transformations which modify data types and ways used for accessing the data. This type of obfuscation is divided into three subcategories [3].

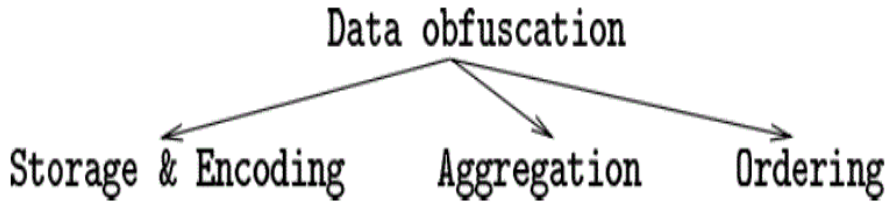

Figure 3 Data Flow Obfuscation Categories.

Storage and encoding obfuscation*:* Storage obfuscations modify the way in which data are stored in the memory, for example: changing data scope from local to global, or from local to elements of object. Encoding transformations influence the way of data interpretation, for example: array indexing as shown on Figure 4.

Before transformation

```
int i = 1;
while (i < 1000){
   ... A[i] ...;
   i++;
}
```

After transformation

```
int i = 11;
while (i < 8003){
   ... A[(i-3)/8] ...;
   i+=5;
}
```

Figure 4 Storage and Encoding Obfuscation Principles.

1. **Aggregation obfuscation:** This category contains data joining and partitioning transformations, for example: concatenation of multiple arrays to single array or encoding boolean value into two integers and their comparison.
2. **Ordering obfuscation:** Contains transformations which modify the ordering of data structures, for example: reorder instance variables and function declarations, reorder methods, reorder arrays.

C. Control flow obfuscation: Contains transformations that attempt to obscure the control flow of the source application. This type is divided into four subcategories based on the operation they perform, see Figure 5.

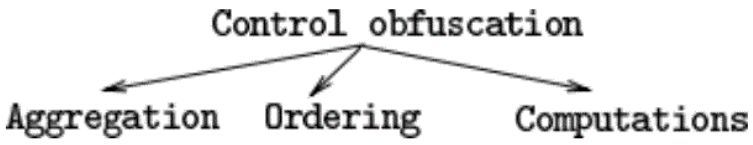

Figure 5 Control Flow Obfuscation Categories.

1. **Aggregation obfuscation:** The transformation modifies the grouping in which the program commands are located. For example, replacing a function call with command from the function body, this technique is called method inlining. Applying this adjustment to all calls will give us a very long sequence code. This technique of obfuscation, according to the authors of the publication [3], is highly resistant because it is a totally one-way transformation, but it is possible to create a tool for searching recurring statement sequences, which can reveal the use of this type of transformation. There is also the method outlining approach that searches for recurring code segments and replaces them with the calls of artificially created functions, but with this method, we need to make sure that the function is called multiple times [4].
2. **Ordering transformation***:* The transformation adjusts the order in which the commands are executed. For example, we can refer to the flipping cycle direction, or the unpacking cycle when the first and last iteration is different from the others, or the separation of even and odd iterations. This also includes inserting of conditional or unconditional jumps [5]. On Figure 6 are shown some basic examples of ordering transformation.

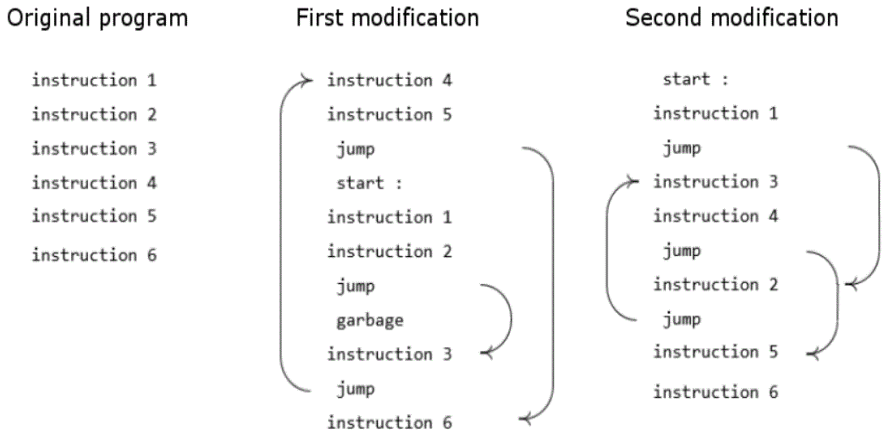

Figure 6 Control Flow Ordering Transformation Examples.

3. **Computation Obfuscation:** Transformation hides the real sequence of commands execution. This goal can be achieved by inserting a dead or blank code, which is a command that does not affect program functionality, such as NOP (No Operation) in Assembler [6]. The dead code can also consist of a more complex block of instructions that will never be executed. We can see some dead code examples in table 1.

Instruction	Semantic
ADD Reg,0	Reg ← Reg + 0
MOV Reg,Reg	Reg ← Reg
OR Reg,0	Reg ← Reg | 0
AND Reg,-1	Reg ← Reg & -1

Table 1 Dead Code Obfuscation Examples

D. **Preventive Obfuscation***:* Unlike the previous two methods, this type of obfuscation targets its modifications on analyzing tools and tries to make de-compilers and de-obfuscators not work properly. This type of obfuscation is divided into two categories.

1. **Inherent Obfuscation:** This kind of transformation focuses on the basic methods used by de-obfuscators, and tries to exacerbate their effectiveness. For example, when rotating a cycle, a fictitious data dependence is added to its body, and hence the obfuscation of the control flow execution is more difficult to remove.
2. **Targeted Obfuscation:** It consists of using transformations designed to degrade the performance of specific analytical tools. If we know the analytical methods of a particular tool, we can incorporate structures into the code, that attack the errors in these methods and thus, to a certain extent, or even completely disable the given analytical tool.

III. EXISTING OBFUSCATION SOFTWARE

Our goal is to make a tool for automatic C code obfuscation. In order to fulfill this task, we have analyzed existing solutions. In their analysis, we looked at what kind of obfuscation they provide, how configurable are they, how difficult is their installation, if they are available for different platforms and we also looked at their usability.

A. COBF 1.06

The program whose main goal is to deter a person from reading and understanding the source code [7]. It accomplishes this goal by removing comments, removing the code formatting, and also renaming variable and function identifiers. The author is Bernard Baier and the latest version of this program 1.06 was released in 2006. The program is distributed as a freeware / shareware. There are executable binary files for Windows as well as files and instructions needed for compiling it for other platforms. COBF 1.06 is used as a command line tool and offers a variety of configuration options using the input parameters. Input parameters are divided into three groups: basic parameters, output parameters and debug parameters. Using the basic parameters, you can set the path to the library files you use, edit the list of mapped identifiers, set the use of custom identifiers, define a set of non-obfuscated identifiers, specify the output folder, and also set the script that calls the preprocessor. Using output parameters, it is possible to set the program to leave the original filenames after obfuscation, to stack all the output files into one, or not to perform the obfuscation of strings because string characters are standardly encoded into their hex ASCII values. Thanks to the debugging parameter, you can see comments on the obfuscated identifiers, add their original names to the identifiers, keep the temporary files, and also turn on the ”filtering mode” and thus turn off the obfuscation. Thanks to all input parameters, we can transform programs with multiple header or library files, selecting which of these files will be obfuscated, and also which variables, macros, or features will not be obfuscated. It is also possible to specify new names of identifiers thanks to the mapping list, which we can edit anyhow.

B. Snob

Snob - Simple (or Stupid) name obfuscator as described by author [8], is a program designed to obfuscate names, titles and identifiers. Snob is available for the Windows operating system in the form of a runtime file. No installation is needed, this file is used as a command line tool with two parameters, the first one is input folder with files that we want to obfuscate and the second one is the output folder. This program is a tool for automatic obfuscating of any language if we have a defined configuration file for it. This file contains the rules for obfuscation defined using regular expressions. Using these rules, it is possible to set the strings, which has to be obfuscated, it is also possible to specify the strings, which must be left in the original state and also to define a set of language-specific symbols and keywords. We can find configuration file for C language and also configuration files for other languages.

C. Scripts Encryptor

The main purpose of the program is to tangle and obfuscate the source code written in ASP, HTML, JavaScript and VBScript. It also offers the ability to encode publicly available web files such as HTML, JavaScript or CSS, as well as minimizing C/C++ and JavaScript code. This program is a commercial product that is only available for the Microsoft Windows operating system and provides support for all new releases of this operating system. Unlike previous tools, it provides a graphical user interface, while the previous ones are only used by the command line. On the website [9], we can find a well-made documentation that describes all the functionality, however it does not provide much documentation for C code obfuscation, because the only available editing feature is to minimize it.

IV. SOLUTION DESIGN

From previous analysis of existing tools for automatic obfuscation, we decided to make a tool that provides both data flow and control flow obfuscation, which will be configured using graphical user interface to ensure easy and intuitive controlling.

A. Graphical user interface

In order to achieve simplicity in use, individual inputs and algorithm settings will be customized using generally known components that widely used in web browsers. These components will include radio buttons, check boxes, buttons and text inputs.

B. Data flow obfuscation

The data flow obfuscation will consist of renaming identifiers, reordering the function and variable declarations, tangling of numeric values, and encoding strings into human meaningless form. The program will provide the ability to specify an input mapping file to identify specific pairs consisting of original identifier and their obfuscated form. It will also be possible to specify the list of identifiers, which will remain in their original state. To preserve the functionality of the code, it is important to leave the original names of the identifiers derived from external library files, the so-called system libraries. We can only edit identifiers that are declared directly in the input file. Functions and variables declarations will always be reordered randomly.

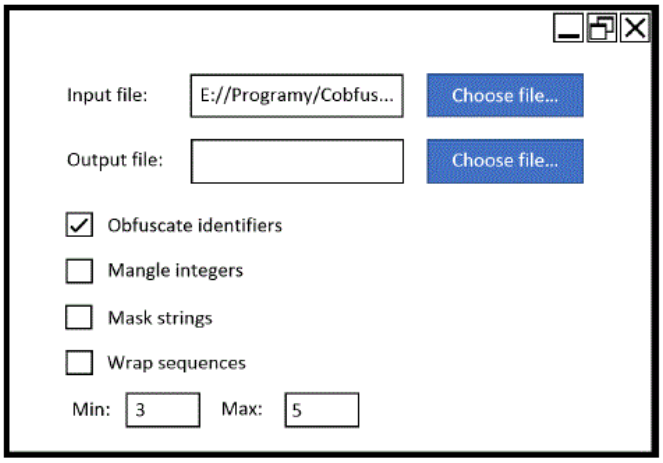

Figure 7 Prototype of Graphic Design Interface.

1. Integers Mangling*:* Integer values will be rewritten to expressions consisting of addition and subtraction, some values will be rewritten into their hexadecimal representation. The example can be seen on the Figure 8.

```
int a = 15;   ➞   int a = (0x27735 - 360050) + 0x3074c;
```

Figure 8 Integers Mangling.

2. Strings Encoding: C language allows using special symbols in strings, these special symbols begin with the prefix of the ”n” slash followed by one or more additional characters. One of the special symbols is the hexadecimal shape of the letter, for example: ”nx4d”, this symbol is translated into a character representing the value ”4d” from the ASCII table. This allows us to use a hexadecimal value symbol to mask individual letters in strings. The example is in the picture 9.

```
printf("Ahoj!\n");   ➞   printf("\x0041\x0068\x006F\x006A!\x0a");
```

Figure 9 String Encoding Example.

C. Control flow obfuscation

Obfuscation of the control flow will consist of wrapping statement sequences into the body of newly created IF, FOR or WHILE statement. For alternatives, we flip conditions and modify the name of the control variable. For cycle commands, we modify the execution conditions and change the increment to decrement, and vice versa.

1. **Wrapping sequences:** The user will be able to specify the interval of the number of commands in the sequence that will be needed for the wrapping. Based on the range of this interval, the target number of commands will be randomly selected, after which the sequence is wrapped. User can also use the settings to specify which of the commands will be used for wrapping. If the user selects more than one option, the wrapping statement will be randomly selected from this set [12].
2. **Flipping condition in alternatives***:* In the case of an alternative command, the evaluation condition is negated and then commands are exchanged between the bodies of the blocks. Example can be seen on the Figure 10 [12].
3.

```
if (a == 10) {
    a++;
} else {
    a--;
}
```

```
if (!(a == 10)) {
    a--;
} else {
    a++;
}
```

Figure 10 Flipping Conditions.

V. SOLUTION IMPLEMENTATION

For implementation, we chose the Java programming language and the IntelliJ IDEA development environment. We have chosen this because we know this environment very well and it provides a full support for writing programs in Java. As project manager, we chose Apache Maven, mainly because it's managing dependencies from external libraries.

A. C Code Parser

If we want to modify the source code written in C, we need to transform it into a form in which it is easy to navigate, analyze and modify it. The most appropriate solution is transformation into an abstract syntax tree. We used the Pycparser tool[10] for the transformation. It is a tool written in python and is designed to parse the C language. Among the examples of uses that the author attached to the distribution is also the conversion from the C language code to the abstract tree. For our needs, we have slightly modified these scripts, and we added functionality to load the input from the file and then write the result to the file. We did this because we will have to call them from our application using Java, and with the arguments of these scripts we will be able to send the paths to the required files. The only downside of this tool is the need for a Python interpreter that is used to call transformation scripts. For proper functionality, user must enter the path to this interpreter.

B. Processing of Header Files

Almost every source code written in C contains directives for accessing library files for preprocessor. The contents of these files are also definitions of new types and functions that can be used in the code. In order to successfully process the code using Pycparser, it is necessary for this tool to know all of these definitions. Pycparser uses cpp, a C preprocessor for processing these library files. As a result, this tool is an essential part of its use.

C. Graphic User Interface

Obfuscator is controlled through the user interface (see picture 11), that we have implemented using components from the JavaFX library. "JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms." [11].

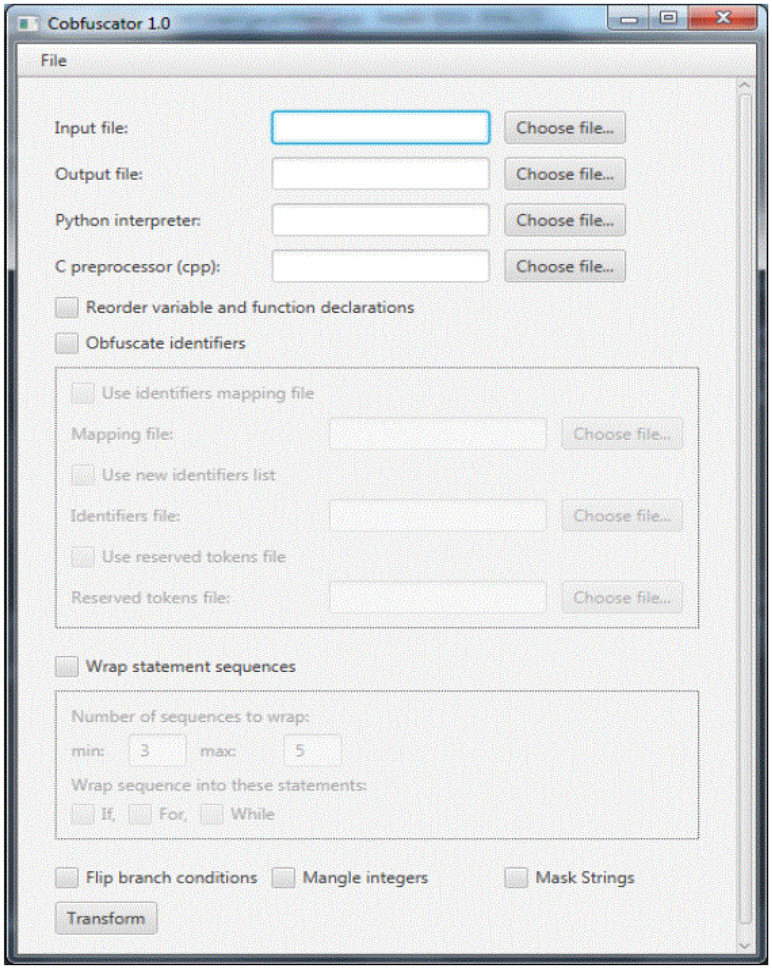

Figure 11 Graphic User Interface.

D. Obfuscator Tool Components

In order to gain better understanding of program functionality, we decided to divide it's functionality into the following components:

- **Main** - The main module secures running of the entire application, initializes other classes, and includes functionality related to configuration and user interface control. Responds to user inputs and adjusts the obfuscation transform settings. It also carries out the initialization actions necessary for a correct transformation.
- **Files Handler** - Large amount of input algorithms are loaded from files, and some files are also generated when implementing the algorithm, that is why we decided to create a custom module for them. This module serves for storing, editing, analyzing and also making these files available for other modules.
- **Tree Handler** - This module contains the obfuscate transformation settings and also includes the functionality associated with its execution.
- **Utils** - A helper module that contains methods for obtaining random numbers and strings used to create new identifiers. It also contains the methods needed to mask the strings.

VI. CONCLUSION

We have succeeded in creating obfuscation tool consisting of simple components that use external module functionality for their operation. These external modules are developed by a strong community of enthusiasts who are involved in editing using publicly available repositories. A great advantage of the output solution is also the use of code transformation to the abstract syntax tree, which represents a structure that every programmer knows very well and therefore this tool is a good basis for eventual editing and upgrading.

ACKNOWLEDGEMENTS

This work was supported by the project 003TUKE-4/2017 Implementation of Modern Methods and Education Forms in the Area of Security of Information and Communication Technologies towards Requirements of Labour Market. The project is being solved at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice.

REFERENCES

[1]. C. COLLBERG, C. THOMBORSON, and D. LOW, A Taxonomy of Obfuscating Transformations. The University of Auckland - Department of Computer Science, 1988, technical Report 148. [Online]. Available:http://profs.sci.univr.it/_giaco/download/Watermarking-Obfuscation/Obfuscation%20Taxonomy.pdf

[2]. C. LINN and S. DEBRAY, Obfuscation of executable code to improve resistance to static disassembly Computer Security Symposium (CSS '03) pp. 290-299, 2003.

[3]. C. COLLBERG and C. THOMBORSON, Watermarking, tamper-proofing, and obfuscation tools for software protection. IEEE Trans. Software Eng. 2002, pp. 735-746., 2002.

[4]. T. OGISO, Y. SAKABE, M. SOSHI, and A. MIYAJI, Software obfuscation on a theoretical basis and its implementation. IEEE Trans. Fundamentals, E86-A(1),, 2003.

[5]. M. UCHNAR, Hybrid system for malware detection. SCYR 2017. - Koice : TU, 2017 S. 124 - 125. – ISBN 978-80-553-3162-1, 2017.

[6]. N. ADAM, B. MADOS, M. CAJKOVSKY, J. HURTUK, and T. TOMCAK, Methods of the Data Mining and Machine Learning in Computer Security. Acta Electrotechnica et Informatica. Year 14, n. 2 (2014), ISSN 1335-8243 p 46-50, 2014. [Online]. Available: http://aei.tuke.sk/papers/2014/2/08 Mados Adam.pdf

[7]. COBF 1.06. [Online]. Available: http://www.plexaure.de/cms/index.php?id=cobf

[8]. Snob - Simple name obfuscator. [Online]. Available: http://www.macroexpressions.com/snob.html

[9]. Scripts Encryptor. [Online]. Available: http://www.dennisbabkin.com/screnc/

[10]. Pycparser - Complete C99 parser in pure Python. [Online]. Available: https://github.com/eliben/pycparser

[11]. What is JavaFX? [Online]. Available: http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[12]. F.A.P. Petitcolas, R. Andersson, J. Kuhn and G. Markus, Information Hiding – A Survey, In: *Proceedings of the IEEE, special issue on protection of multimedia content*, 87(7):1062-1078, July 1999.

*Ján Hurtuk1. "Simple Obfuscation Tool for Software Protection." Quest Journals Journal of Software Engineering and Simulation 3.7 (2017): 14-21.