International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:10, 2007

Obfuscation Studio Executive

Siarhei Petryk, and Vyacheslav Yarmolik

***Abstract*—**New software protection product called "Obfuscation Studio" is presented in the paper. Several obfuscating modules that are already implemented are described. Some theoretical data is presented, that shows the potency and effectiveness of described obfuscation methods. "Obfuscation Studio" is being implemented for protecting programs written for .NET platform, but the described methods can also be interesting for other applications.

***Keywords*—**Coupling, obfuscation, predicates, renaming.

I. Introduction

ONE of latest software innovations is the platforms that use compilation of source code to the assemblies in some intermediate languages, not in native code. The examples of such platforms are Java (uses Java byte-code) and .NET (uses Microsoft Intermediate Language). Such approach allows to get a lot of advantages: allows JIT (Just-In-Time) compilation of assemblies to the native code that suites particular hardware environment, improves extendibility and others. At the other hand, it is very easy to decompile such assemblies and analyze used algorithms. This weakness is used by reverse engineers: they can use non-licensed algorithms, or remove watermarks and fingerprints from the programs.

The method that is used to prevent attacks described above is obfuscation [1]. At the moment there is no strict definition of the term obfuscation, the weak definition is the following: obfuscation is a process that converts program to the functionally equivalent one but that is harder to attack by reverse engineering methods [2].

Some software companies develop obfuscators, but most of them implement only primitive methods that are not based on the theoretical results or even are not practically analyzed on real or sample applications [3].

The work-in-progress obfuscator "Obfuscation Studio" is presented in this paper. It implements several methods of obfuscation that were discussed in literature [2, 3, 4]. Moreover, the paper presents the results of practical investigation of described methods. These results show its effectiveness and advantages.

Manuscript received October 14, 2005.

Siarhei Petryk and Vyacheslav Yarmolik are with the Belorussian State University of Informatics and Radioelectronics, Minsk, Belarus (e-mails: siarhei_petryk@tut.by, yarmolik@bsuir.unibel.by).

Vyacheslav Yarmolik is with Bialystok University of Technology, Poland (e-mail: yarmolik@ii.pb.bialystok.pl).

II. Obfuscation Studio Architecture

The "Obfuscation Studio" architecture is presented on Fig. 1.

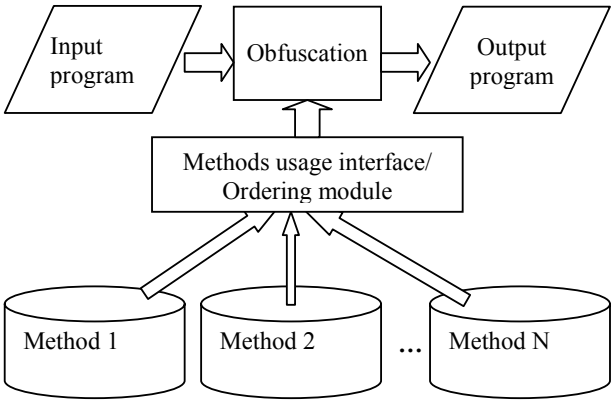

Fig. 1 Obfuscation Studio Architecture

The main module is called "Obfuscation". It implements functionality of performing transformations. The libraries with such transformation are dynamically linked to the main module through the interface "Methods usage interface/Ordering module". This module analyses transformation libraries and creates an order of transformations.

Input data are files with source code of the program. Output data is a program after transformations.

Three obfuscation modules are currently implemented:

1. Entities renaming (lexical method);
2. Building predicates in the code;
3. Increase modules coupling.

All these transformations are described in details below.

III. Lexical Method of Obfuscation

As it is mentioned above lexical method means entities renaming [5]. Here it is as an example the sample procedure that will be shown obfuscated later:

Listing 1. Source code before obfuscating

```
private Hashtable getFrequency(string fileName)
{
        int readByte;
        FileStream fs = new FileStream
                (fileName,FileMode.Open,FileAccess.Read,FileShar
e.None);
```

```
        Hashtable frequency = new Hashtable();
        while ((readByte=fs.ReadByte())!=-1)
        {
          if (frequency[readByte]!=null)
                frequency[readByte]=(int)(frequency[readByte]) + 1;
          else
                frequency.Add(readByte,1);
        }
        fs.Close();
      return frequency;
  }
```

A simple cryptographic procedure is presented in Listing 1. It analyzes input file and returns the hash-table that contains all the symbols from input file and their frequencies.

After transformation with "Obfuscation Studio" the procedure was converted to the following:

Listing 1. Source code after obfuscating

```
private Hashtable ll1lll11(string ll1lll1l)
{
        int ll1lllll;
        FileStream ll1ll1ll = new FileStream
                (ll1lll1l,FileMode.Open,FileAccess.Read,FileShare.N
one);
        Hashtable ll1ll1l1 = new Hashtable();
        while ((ll1lllll=ll1ll1ll. ReadByte())!=-1)
        {
          if (ll1ll1l1[ll1lllll]!=null)
              ll1ll1l1[ll1lllll]=(int)(ll1ll1l1[ll1lllll]) + 1;
          else
              ll1ll1l1.Add(ll1lllll,1);
        }
        ll1ll1ll.Close();
        return ll1ll1l1;
}
```

It is evidently that the code became more complex for reverse engineers, but computer does not see the difference that means that the program will operate with the same speed.

One of the action items that can be performed for lexical method of obfuscation is the estimation of obscuring potency of the investigated method. We suggest defining the following dependencies to get the value of obfuscation potency:

Characteristics A: dependence between the number of different entities names in a program and the number of different words in a program;

Characteristics B: dependence between the overall number of entities names and overall number of words;

Characteristics C: dependence between the overall length of all entities names and the length of the entire program.

The investigation of mentioned was performed for the programs written on such languages as C, C++, C# and Java. Below we present only investigation result for one of the languages (C#) because the results for other languages are very close to the listed below [3].

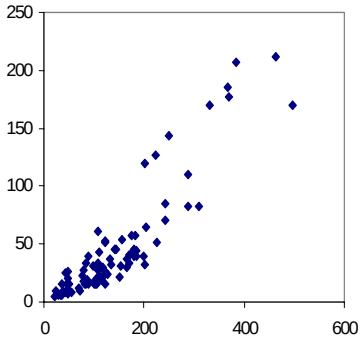

Fig. 2 Characteristics A for C#

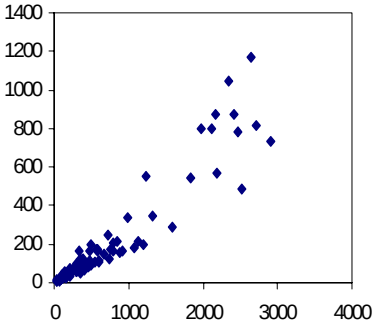

Fig. 3 Characteristics B for C#

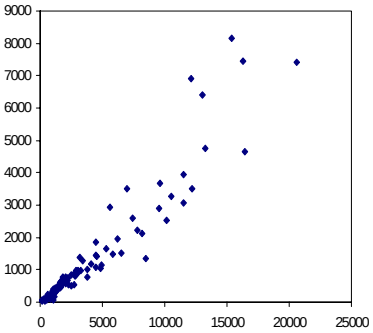

Fig. 4 Characteristics C for C#

We can get the ratio of the values that characterize the entities and the entire program; this shows the part of a program that is taken by the entities names. The percentage equivalent of that ration is 15-55%, this part of program can be changed and obscured by the lexical obfuscation method. By the way, this part of code can be used for building watermarks in the source file and possibly for some other purposes [3].

The disadvantage that should be mentioned while discussing lexical method is the simplicity of performing reverse actions. Those actions will not bring back sensible names, but will make it easier to understand the logic of a program.

IV. Obfuscation with Predicates

The only lexical method of obfuscation will not give any guaranties of programs protection. Therefore a lot of other approaches were presented that change the flow-graph of a program. One of the approaches that is implemented for "Obfuscation Studio" is the module that builds predicates into source code [6, 7]. The following examples will help to understand the idea:

Listing 3. Source code before obfuscating

```
public Class1 ( )
{
   d = a + b ;
   e = d + c ;
   f = a + c ;
   g = e + f ;
}
```

Listing 4. Source code after obfuscating

```
public Class1 ( )
{
   if (TrueFlag)   if ( TrueFlag )  d = a + b ;
   if (!FalseFlag)
   {
      if ( TrueFlag )  if ( !FalseFlag )  e = d + c ;
   }
   else
   {
      if ( TrueFlag )  if ( !FalseFlag )  e = d + c ;
   }
         if (TrueFlag)  if ( TrueFlag )
      if ( TrueFlag )  f = a + c ;
   if ( !FalseFlag )
   {
      if ( !FalseFlag )  g = e + f ;
   }
   else
   {
      g = e + f ;
   }
}
```

The obfuscating module algorithm is the following:

1) Procedures bodies are being divided into elementary parts (operations);
2) Obfuscator chooses the random set of such operations and transforms it with one of the following rules:
   1. Op → if (PrT) Op;
   2. Op → if (Pr?) Op; else Op;
   3. Op → if (PrT) Op; else Op';
   4. Op → if (not PrF) Op;

   where Op – a set of operations,

Op' – a set of operations that which can replace the original set,

PrT – predicate that is always true,

PrF – predicate that is always false,

Pr? – predicate that can be either true or false.

3) Initial code is being replaced with transformed one.

To determine the effectiveness of using presented method of obfuscation we suggest the following metrics:

1. predicates diffusion of a program;
2. average predicates nesting level;
3. complexity of selecting expressions;
4. symmetry of predicates distribution.

The variable parameter for presented method of obfuscation is the number of passes on the same part of code. Below there are diagrams that show the dependencies between some mentioned metrics, the length of the program, the execution time and the number of passes.

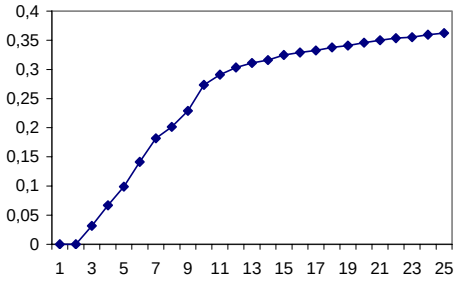

Fig. 5 Dependence between the predicates diffusion and the number of passes

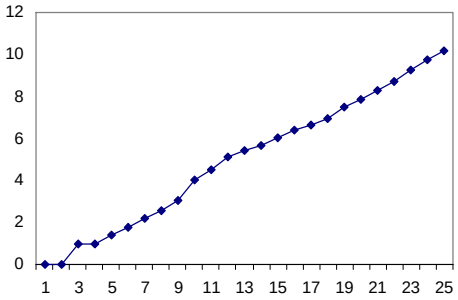

Fig. 6 Dependence between the average predicates nesting level and the number of passes

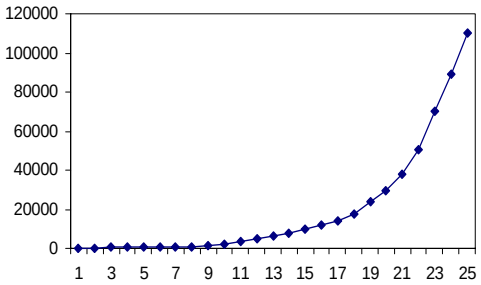

Fig. 7 Dependence between the length of the program and the number of passes (in bytes)

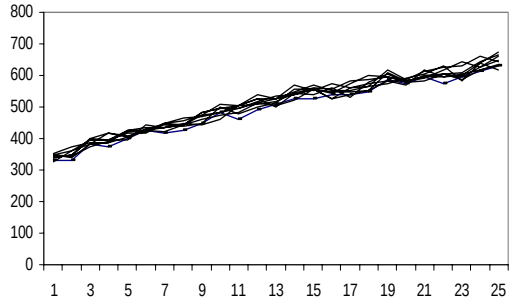

Fig. 8 Dependence between the execution time (in milliseconds) and the number of passes (10 measurements were performed)

As it is shown on the Fig. 7 the program length grows exponentially that is undesirable. That's why it is needed to resolve the problem of optimal obfuscating parameters (will be discussed later).

V. Increase Moduls Coupling

The latest implemented obfuscation library brings to "Obfuscation Studio" the ability of using the method of obfuscating by increase program modules coupling [8].

The main idea is the following: module is more complex if it interacts with more other modules [9]. The obfuscating algorithm idea looks like the following: the obfuscator goes through all the modules, gets the random number of operators from each module and builds it in the other modules. Natural restrictions for this algorithm are connected with the rules of object-oriented programming.

The metric that measures the coupling is called CBO (Coupling Between Objects) that determines the number of classes with which interacts current module [10].

For testing purposes we used medium application that consists of 17 classes, that performs some operations with financial data. The dynamic of CBO changing and program length growing is shown on the Fig.9 - Fig. 10.

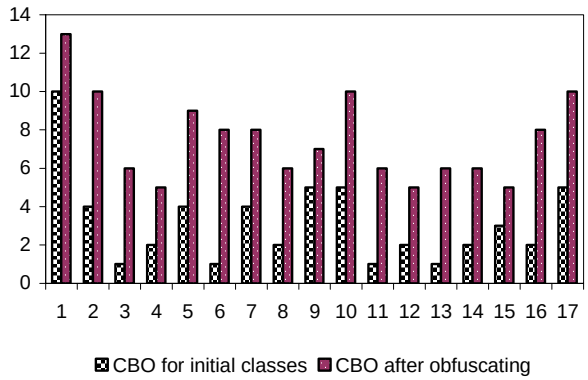

Fig. 9 CBO growing

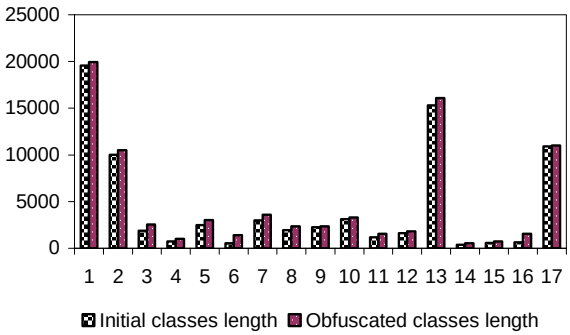

Fig. 10 Length growing

The investigation of sample application showed that the average value of CBO metrics for all the classes has grown form 3.18 to 7.53, the ratio is 2.37. The ration for growing program length is 1.10 that is less then ratio of CBO growing. This means that the program complexity grows faster then its length.

VI. Optimal Obfuscation Quality

It is well known that the obfuscation methods that transform the programs flow-graph cause the programs length and execution time growing that is desirable in most cases [8]. That why we built special algorithms in "Obfuscation Studio" that find the optimal obfuscation parameters. It uses one of the resolving optimization problems methods - multiplicative convolution. General formula for this method is the following:

$$E = \frac{\prod_{j \in J\uparrow} k_j^{\delta_j \times \alpha_j}}{\prod_{j \in J\downarrow} k_j^{\delta_j / \alpha_j}} \quad (1)$$

where k_j – the value of j criteria (metric),

δ_j – factor that makes all values normalized,

α_j – importance factor of j criteria (from 0 to 1 as defined by the expert),

$J\uparrow$ – criteria that are to be maximized,

$J\downarrow$ – criteria that are to be minimized.

Maximization is required for the metrics values and the minimization is required for program length and execution time.

References

[1] *Reverse Engineering Wizard.* Microsoft .NET Framework SDK Tool Developer's Documentation. Microsoft Corporation, 2001.

[2] Christian Collberg, Clark Thomborson, Douglas Low. *A Taxanomy of Obfuscating Transformations.* Technical Report 148, Department of Computer Science, University of Auckland. – 1997.

[3] S. Petryk, V. Yarmolik, *Investigation of lexical method of obfuscation* . Informatics. Vol. 3, 2004, pp. 58-66.

[4] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S.Vadhan and K.Yang. *On the (Im)possibility of Obfuscating Programs.* Electronic Colloquium on Computational Complexity (ECCC). – Vol. 8(057) – 2001.

[5] Prashant Shah. *Code Obfuscation For Prevention of Malicious Reverse Engineering Attacks.* ECE 578, Computer and Network Security. – 1998.

[6] S. Petryk, *Obfuscating by building predicates in.* Proc. 10th Russian conference in Ryazan. 2005. pp. 82-84.
[7] Douglas Low. *Java Control Flow Obfuscation.* University of Auckland. – 1998.
[8] Sergei Petrik, Vyacheslav Yarmolik. *Obfuscation by influence the modules coupling.* Automatic Control and Computer Sciences, to be published.
[9] Arun Lakhotia, "Rule-based Approach to Computing Module Cohesion". Proceedings 15th International Conference on Software Engineering, Baltimore. – 1993.
[10] Shyam R. Chidamber, Chris F. Kemerer, A Metrics Suite for Object Oriented Design. *IEEE Transactions on software engineering*. – vol. 20. – 1994.