<table>
<thead>
<tr>
<th>Title</th>
<th>Obfuscation techniques for Java source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Nguyen, Dinh Phuc</td>
</tr>
<tr>
<td>Citation</td>
<td>Nguyen, D. P. (2014). Obfuscation techniques for Java source code. Student research paper, Nanyang Technological University.</td>
</tr>
<tr>
<td>Date</td>
<td>2014</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10220/26030">http://hdl.handle.net/10220/26030</a></td>
</tr>
<tr>
<td>Rights</td>
<td>© 2014 The Author(s).</td>
</tr>
</tbody>
</table>
Obfuscation Techniques for Java Source Code

Nguyen Dinh Phuc
School of Computer Engineering

Assoc Prof Ng Wee Keong
Mr Vivek Balachandran
School of Computer Engineering

Abstract – Nowadays, software piracy becomes one of the headaches of any software companies. There are many techniques to protect a software from being copy, reverse-engineered or modified before delivering to user. Some of them can be listed out such as watermarking, embedding decryption keys in hardware, tamper-proofing, fingerprint code, virtual machine… Obfuscation is a very cheap and can be conducted at multiple levels of complexity and in multiple stages of a program. This technique secures the software by making it obscure and difficult for hackers to understand the function of codes and reverse-engineer. This project focuses on obfuscation techniques at the highest level of a program: source code, namely variable and class renaming, array restructuring, including array splitting, array merging, array folding and array flattening. The project uses Java programming language and Java Development Tools in Plug-in environment of Eclipse.

Keywords - Obfuscation; Java programming language; Software protection, Abstract-syntax tree, Parser.

1 INTRODUCTION

Normally, a program is compiled to machine codes before being run. Most of the supporting information is stripped off when the program is compiled [1]. Although decompiling a program is difficult, it is still possible. The lower level of a program to be decompiled, the harder the hackers can crack. Unfortunately, there is no way to perfectly protect a software from being cracked. The only way is to make the reverse-engineering time-consuming and confusing. Obfuscation can be a good, effective and cheap approach. Obfuscation can be used in various level of a program: source codes, binary, byte code, machine code. Moreover, one obfuscation technique can be applied multiple times until it reaches desired level of complexity. An obfuscator is considered successful if it makes the reverse-engineering costly in time and effort. Obfuscation can be classified into four types:

- Design obfuscation: class merging, splitting.
- Data obfuscation: array restructuring.
- Control obfuscation: hidden control flow info
- Layout obfuscation: variable renaming, deleting comments.

This project works on layout and data obfuscation. Variables can be renamed to a randomly generated name with no sense. Arrays are detected and restructured. Array can be split into two new arrays from one original array. It is also possible to merge two arrays into only one. Moreover, one dimensional array can be folded to two or three dimensional array depending on how desired level of obfuscation to be achieved.

2 BACKGROUND KNOWLEGDE

2.1 ECLIPSE PLUG-IN

Eclipse provides an extensible environment where developers can write their own plug-in and install for the Integrated Development Environment (IDE) [2]. This project makes use of Eclipse plug-in environment to implement a simple obfuscator for Java program. After developer has already finished their program, they simply need to click an icon on the taskbar of Eclipse IDE. The program will be obfuscated.

2.2 JAVA DEVELOPMENT TOOLS

Java Development Tools (JDT) was developed by Eclipse. JDT allows users to write, compile, test, debug and edit programs written in Java programming language. It adds a java project nature and Java perspective to Eclipse Workbench as well as number of views, editors, wizards, builders, code merging and refactoring tools [3]. In addition, it provides an effective parser to parse source codes and build an abstract-syntax tree (AST). The generated AST allows developers to monitor projects, variables, classes, packages and modify source code directly.

2.3 OBFUSCATION PRINCIPLES

An obfuscator is good if it costs hackers a longer time to decompile a program than to write that piece of codes. Obfuscation transformations were first introduced by Collberg [4].

\[ P \xrightarrow{T} P' \]

Where P is the original program, P’ is the obfuscated program after applying transformation T. The necessary condition is that P and P’ have the same behaviors, but P’ is much more obscure than P. In fact, P’ is usually slower than P or using more memory than P because it is more complex. Complexity of P’ should be at most polynomial larger than that of P and time taken the
hackers to recover P from P’ should be longer than time to develop P from scratch [5].

3 IMPLEMENTATION

The project works on identifiers renaming and array restructuring. Firstly, the project is parsed using JDT to build an AST. Secondly, the plug-in traverses all the AST Nodes and modify where necessary. Lastly, the project is written back to files.

3.1 CREATE PROJECT AST

The project is first checked to make sure that it is a java project. All the packages and Java classes are scanned one by one to generate the corresponding Compilation Unit. The obfuscator uses this Compilation Unit to create AST and then accept the AST Visitor to traverse all AST Nodes.

3.2 IDENTIFIERS RENAMING

The candidates for identifier renaming:
- A package
- A class or interface
- A nested type
- A field
- A method
- A parameter (of a method, constructor or exception handler)
- A local variable

This first version of obfuscator only supports package, class, field, method renaming. It makes use of a dedicated method to generate random name for each identifier detected.

JDT provides a Rename Support class which helps to refactor an identifier in the scope of the whole project. The result is displayed as below:

```java
package packageA;

public class ClassB {
    private int a;
    private double b;
    private String c;
    private int[] array1;
    private int[] array2 = new int[13];
    private ClassA e;
}

public ClassB(int a, double b, String c, ClassA e) {
    super();
    this.a = a;
    this.b = b;
    this.c = c;
    array2 = new int[12];
    this.e = e;
}

public int getA() {
    a = array2[6];
    array2[6] = 10;
    for (int i = 0; i < array2.length; i++)
        System.out.println(array2[i]);
    return a;
}
```

In future work, the obfuscator will be equipped with new capabilities of renaming local variables, nested type and method parameters.

3.3 ARRAY RESTRUCTURING

To get the arrays declaration in a class, all the field declaration nodes need to be visited and stored in a list if it is of array type. The list is then used to apply modifications such as array splitting, array merging, array folding and array flattening. The first version of the obfuscator only supports array splitting. The other transformations can be developed similarly.
After detecting an array declaration, the obfuscator splits the original array into two new arrays. The two new arrays are stored in a list, which then applies the changes to corresponding AST nodes where the array is referenced. The first half elements are present in the first array, the last half are in the second array.

- In field declarations.
- Inside constructor.
- Array access.
- Array creation.

An AST visitor is created to visit all the statements one by one. In each statement, it checks the nodes rooting from that statement whether it is referencing to the old array. Another AST visitor created to traverse array access nodes that belong to the statement and modify it. The obfuscator analyses the index of array access node and adds an if-else statement where applicable:

- If index is a known integer, compare index with the length of first new array.
- If smaller, remain the index, rename the old array to first array.
- If greater, rename to the name of second array, the index is now: index – length of first array – 1.
- If index is an expression, an if-else statement needs to be added to the source codes to compare the index expression and length of first array.

There is another method to split array. The first array contains all the elements with odd position and the second array refers to elements with even index. If so, the if condition is set to modulus

The experiment before and after running obfuscator as below (the renaming capabilities of other variables were disabled for ease of reading, the old arrays are kept for reference):

```
public class ClassB {
    private int[] CFKqGMSFlnKpq = new int[6];
    private int[] DgFKEcGdIVHAFpnpBq = new int[7];
    private int[] YhsEcWNevoVY1xOC;
    private int[] OEFKqQFXy7rVZ;
    private int a;
    private double b;
    private String c;
    private int[] array1;
    private int[] array2 = new int[13];
    private String e;
}
```

```
public class ClassE(int a, double b, String c, ClassA e) {
    super();
    this.a = a;
    this.b = b;
    this.c = c;
    Aqznjwz58gq7wQ = new int[6];
    faceTevKbQGLearwVR = new int[6];
    this.e = e;
}
```

```
public int getA() {
    a = DgFKEcGdIVHAFpnpBq[6];
    DgFKEcGdIVHAFpnpBq[6] = 10;
    for(int i = 0; i < array1.length; i++)
    {
        if (i <= 7) {
            System.out.println(DgFKEcGdIVHAFpnpBq[1]);
        } else {
            System.out.println(CFKqGMSFlnKpq1 - 4);
        }
    }
    if (i <= 7) {
        System.out.println(DgFKEcGdIVHAFpnpBq[1] = 3 * DgFKEcGdIVHAFpnpBq[1] + a);
        else {
            CFKqGMSFlnKpq1 - 6 = 3 * CFKqGMSFlnKpq1 - 6 + a;
        }
    }
    return a;
}
```

```
public int getB() {
    a = DgFKEcGdIVHAFpnpBq[6];
    DgFKEcGdIVHAFpnpBq[6] = 10;
    for(int i = 0; i < array1.length; i++)
    {
        if (i <= 7) {
            System.out.println(DgFKEcGdIVHAFpnpBq[1]);
        } else {
            System.out.println(CFKqGMSFlnKpq1 - 4);
        }
    }
    if (i <= 7) {
        System.out.println(DgFKEcGdIVHAFpnpBq[1] = 3 * DgFKEcGdIVHAFpnpBq[1] + a);
        else {
            CFKqGMSFlnKpq1 - 6 = 3 * CFKqGMSFlnKpq1 - 6 + a;
        }
    }
    return a;
}
```

**CONCLUSION**

This project has kicked off an obfuscator for Java source codes. In the beginning, it only supports identifiers renaming and array restructuring. This obfuscator is implemented as a plug-in in Eclipse and makes use of Java Development Tools to analyze and dynamically
modify source codes. In the very first version, the obfuscator is not really effective because it loops the whole project many times in order to modify source codes.

In the future work, the obfuscator will be improved with new features such as constant hiding, comments and space removal and other type of array transformations. This obfuscator can be extended with class transformations such as class splitting, class merging developed by my partner in another parallel project. Moreover, it may be developed as a stand-alone application rather than a plug-in. Hence, it will not be restricted to Eclipse projects only, but all Java software programs.

ACKNOWLEDGMENT

I would like to thank my supervisor, Associate Professor Ng Wee Keong for giving me this opportunity to do this project and learn the first knowledge of software protection.

I would like to thank Vivek Balachandran, my co-supervisor, for his effort to give me the first knowledge about obfuscation techniques and his guidance during the first stage of the project.


We wish to acknowledge the funding support for this project from Nanyang Technological University under the Undergraduate Research Experience on CAMpus (URECA) programme.

REFERENCES


