*2011 International Conference on Information and Network Technology*
*IPCSIT vol.4 (2011) © (2011) IACSIT Press, Singapore*

# Suggesting Potency Measures for Obfuscated Arrays and Usage of Source Code Obfuscators for Intellectual Property Protection of Java Products

Praveen Sivadasan [1], P.Sojan Lal [2]

[1] Research Scholar, School of Computer Sciences, Mahatma Gandhi University, Kerala, India
[2] Research Guide, School of Computer Sciences, Mahatma Gandhi University, Kerala, India

**Abstract.** Malicious reverse engineering aims on understanding and reusing the functionalities of existing products for the development of other products. It helps the competitors to release the product without much research, development time and investment, thereby causing huge financial losses to the developing organizations. Intellectual property violations due to malicious reverse engineering are serious issues faced by internet java applications, which causes huge financial loss to the product development organizations. Obfuscation is an economical approach of transforming a program to another incomprehensible version, for curtailing down such malicious reversing attacks. Our research is on obfuscating java source codes by obscuring array usages through data transformation and we have developed source code obfuscation frameworks named JConstHide and JDATATRANS. Potency is the degree to which the reader is confused and in this paper, we have suggested metrics for assessing potency of obfuscated arrays and also the application of source code obfuscation tools during software development to add more protection to the proprietary software products.

**Keywords:** Source Code Obfuscation, Reverse Engineering, Intellectual Property Protection

## 1. Introduction

Internet Programming has been gaining more popularity and malicious reverse engineering[9,11] is an intellectual property violation attack faced by java applications, which aims on understanding and reusing the functionalities of an existing product in the development of other products. Such attacks make it easier for the competitors to extract the proprietary algorithms and data structures from Java applications in order to incorporate them into their own programs, to cut down their development time and cost. Decompilation is the process of generating source codes from intermediate byte codes and the semi compiled nature of Java class files make it more amenable to reverse engineering and re-engineering attacks through decompilation. Such cases of intellectual property thefts are difficult to detect and pursue legally.The American Society of Industrial Security (ASIS) has stated that the intensity of intellectual property thefts are escalating and most of the companies spend less than 5% of their budgets for security. ASIS has calculated losses due to theft as 45 billion dollars in 1999 with the latest figure as 150 billion dollars per year. Software obfuscation[1,2,7,8] is a popular approach where the program is transformed into an obfuscated program using an 'obfuscator' in such a way that the functionality and the input/output behavior is preserved in the obfuscated program whereas it is much more difficult to reverse engineer the obfuscated program. Though, obfuscation is a more economical method for preventing reverse engineering, there are 'deobfucators' available to defeat some of the less sophisticated obfuscation strategies. The popular transformation techniques employed for obfuscation are (i) *layout transformation* which makes the structure of the transformed program difficult to comprehend (ii) *data transformation* that obscures the crucial data and data structures (iii) *control transformation* to obscure the flow of execution. The obfuscation can be performed on the source code, the intermediate code or the machine executable code. Source code obfuscation is little hard to perform, because

of various program constructs, and it achieves byte code obfuscation. The effectiveness of obfuscation is usually measured in terms of a) the *potency* that is the degree to which the reader is confused, b) the *resilience* that is the degree to which the obfuscation attacks are resisted and finally c) the *cost* which measures the amount of execution time/space penalty suffered by the program due to obfuscation. The software tools like byte code obfuscators alter byte codes, where as source code obfuscators[3,5,6] obscure source codes. Reverse engineering attacks through decompilation[12] are common with distributed and easily decopilable formats like java class files.

## 2. Array Obfuscation

Data transformation [1,10] is our main area of interest, where array splitting, array folding, array flattening and array merging are the popular array transformation techniques. The split arrays, folded arrays, flattened arrays and merged arrays are referred as restructured arrays or obfuscated arrays. The following shows split arrays A1 and A2 for array access, where even indices elements are stored in A1 and others in A2. The following codes are also referred as restructured codes for split arrays.

```
int A1[5], A2[5] ;
if ((i%2)=0) A1 [i/2] =…….; else  A2 [i/2] =…….;
```

The folded array is a two dimensional array D1, which is shown below,

```
int D1[2][5];
D1[(i-(i%5))/5][i%5]=……;
```

The flattened array, shown below is a one dimensional array E1.

```
int E1[9];
E1[3*i+j]=……;
```

The mered array is a one dimensional array BC,

```
int BC[30];
BC[3*i]=……;
BC[i/2*3+1+i%2]=……;
```

The source code is obscured by considering *restructured arrays* for array usages to harden the index computation. Array indices help locating the array elements and hard index computations toughens locating array elements and thereby makes reverse engineering of source codes hard for the attacker. We call an array to be of significant potency, if it has reasonable difficulty in index computation. Reverse engineering effort is the effort spent on understanding the obfuscated codes and more effort could make reverse engineering tiring for the malicious attacker and could curtail down such attacks. We have developed the framework named JDATATRANS[3] to implement source code array obfuscation, whose key components are are *CoBS* (Classes for oBfuscating Source codes) repository that contain generic classes for various obfuscated array implementations and an *Obfuscator* that obfuscates the Java programs that uses the CoBS arrays.

A sample obscured source code myprog.java using the SplitArray class is shown below,

```
SplitArray<Integer>ar=newSplitArray<Integer> (100000);
ar.setArray(i,(3*i + 1000) % n);
y = ar.getArray(i);
```

For iterations on myprog.java, JDATATRANS tool hides the constants by F() calls and the earlier code is transformed into the following code,

```
SplitArray<Integer>ar=newSplitArray<Integer> (50000*F(49135%24575,12));
ar.setArray(i*(4*F(3059%1535,8)(F(49135%24575,12)*F(35%27,2)+F(33%21,2))),(3*i + 1000)% n);
y=ar.getArray(i*(F(35%27,2)-F(12273%6143,10)));
```

## 3. Potency Measures for Obfuscated Arrays and Applications of Source Code Obfuscators in Product Development

We have made use of Halstead's metric [4] for length and volume of a program based on the number of distinct operators and operands, to propose a metric for the potency of restructured arrays. For a program, Halstead's metric considered the following measurable quantities,

n1=the number of distinct operators

n2=number of distinct operands

N1=the total number of operators

N2=the total number of operands

Vocabulary(n)=n1+n2, where n is the total number of unique tokens from which the program has been constructed

Length(N)=N1+N2

Volume(V) as $V=N \log_2 n$ (the program's physical size)

We suggested measuring the potency of restructured arrays based on the complexity of the array index expressions of the restructured codes. We proposed applying the vocabulary aspect of the Halstead's metrics to the expression, for measuring its complexity. We introduce a terminology called V(M), referred by the *Vocabulary of an expression M* , to express the complexity of any expression. The vocabulary of expression i/2*3+1+i%2 is given by V(i/2*3+1+i%2)=8.The the total score for any restructured codes with distinct index expressions, where $E_i$ represents $i^{th}$ distinct index expression is $\mathbf{Sc_I = \sum V(E_i)}$. If $C_j$ represent the $j^{th}$ distinct conditional expression and let $V(C_j)$ be its vocabulary, then the score is $\mathbf{Sc_K = Sc_I + \sum V(C_j)}$. For the $i^{th}$ index expression containing recursive data hiding function calls, say F(F(F())),the count denoted by $k_i$ is 3. The new score is $\mathbf{Sc_F = Sc_K + \sum k_i}$. Since the operations * / % has higher precedence , we have assumed the precedence factor value for each operator by taking the sum of vocabularies of each operator, ie, O(*)=O(/)=O(%)=V(*)+V(/)+V(%)=3. The operators + , – are considered as normal tokens and each operator is assigned value 1. That is, O(+)=V(+)=1 and O(-)=V(-)=1. The vocabulary calculation based on the precedence factor is V(3*i)=V(3)+O(*)+V(i)=1+3+1=5. Let $SC_P$ denote the scores based on the precedence factors and $SC_N$ the scores ignoring the precedence factors. The following Table1 consolidates the scores for array restructuring techniques based on $SC_N$ and $SC_P$.

| Array Restructuring | $SC_N$ | $SC_P$ |
|---|---|---|
| Array Merging | 11 | 19 |
| Array Folding | 8 | 14 |
| Array Flattening | 5 | 7 |
| Array Splitting | 7 | 11 |

Table 1 Scores based on SCN and SCP

From the Analysis in Table1, among the 4 restructuring operations considered, based on our suggested metrics array merging is found to be more potent among the restructuring techniques. The image filters are being used in java image processing applications and audio filters are used in voice applications, where one dimensional integer arrays are the common data structures used for designing the filters. Some of the image filters are ImageCombiningFilter, MinimumFilter, UnsharpFilter, LightFilter denoted by A, B, C, D respectively. The filters A, B, C, D use 3,3,2,6 integer arrays. The proprietary filters developed in Java, to an extent, can be strengthened and can be protected from reversing attacks by obscuring the arrays. The framework JConstHide adds more obscurity by hiding the constants of the source code and JDATATRANS obscures the arrays. The following Figure1 illustrates the usage of frameworks to add more potency to the java filter codes.

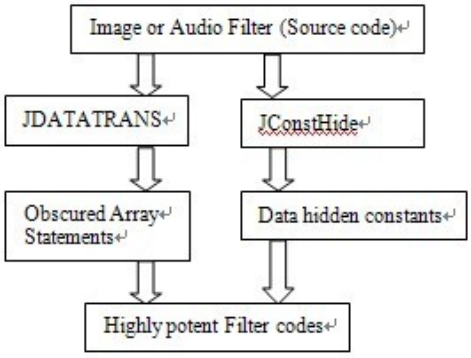


Figure 1 Usage of tools to strengthen filter codes

Every company has an intellectual property to protect which often includes algorithms built right into the software that is sold to customers and the secrecy of such software is an edge to beat their competition in the market. The key threats faced by many of the software product development industries are 1) *Reverse Engineer and Use* 2) *Steal and Sell*. There are two common practices of protecting an intellectual property of a software producer - Legal and Technical methods. '*Reverse Engineer and Use*' could result in reverse engineering and use of logic in future product development, by the competitor or some times by the customer. *'Steal and Sell'* happens during the product development where the developer can comprehend the idea of the modules and sell it to a third party. This is also a serious threat during product development stage, which is a difficult issue to deal. The existing byte code obfuscators can only be operated on error free and syntactically complete source code, but our suggested obfuscators perform obfuscation at any point of time of development of the source code without requiring the compiled version of the code. All java products contains a collection of byte code files and we suggest adding more reverse engineering effort to the decompiled codes by using the existing byte code obfuscators on compiled obfuscated byte codes, prior to selling, in order to strengthen the product from illegal attacks. The following Figure2 illustrates this strength addition procedure.

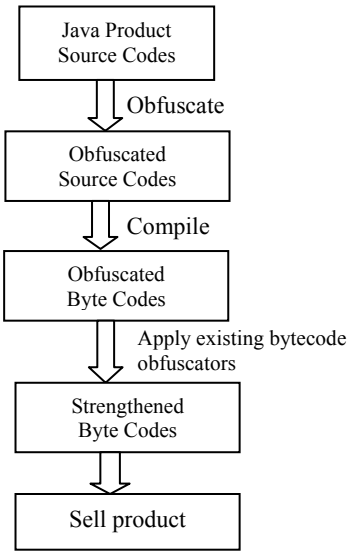


Figure2 Strengthening of byte codes with existing obfuscators

Also, often the java product in development phase could result in software theft by a developer. In Yahoo news on June 23, 2009, news was reported where a software engineer was held for cheating the company for Rs.46.5 crores, by stealing the source code and starting his own company. Such cases of intellectual property thefts are also difficult to detect and pursue legally. Any product P is a combination of modules $M_1$,$M_2$,……$M_n$ and let $M_k$ be a proprietary module of the organization, to be used various product development and its logic not to be exposed to the developers. In this case, the module $M_k$ can be obscured and distributed for incorporating it in the product development. The following Figure 3 illustrates the procedure. The shaded box denotes the crucial module.

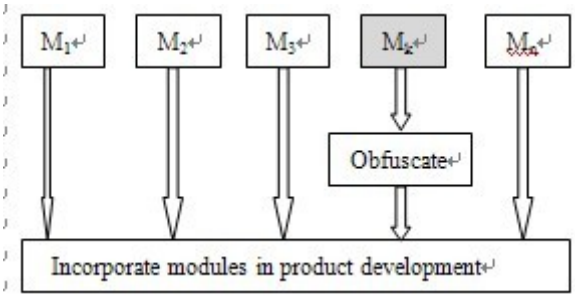


Figure3 Obfuscating crucial module for product development

Similarly, a significant module $M_K$ under development by developer P, in Figure 4(a), has to be distributed to another developer, say Q, for further development. The codes till developed can be obscured to hide its functionality and is supplied for further development, which would limit reverse engineering of codes by Q.

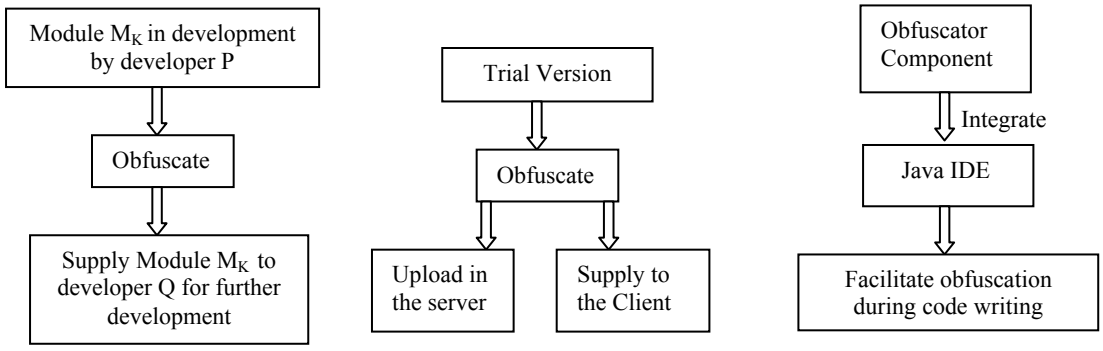


Figure4 (a) Crucial module obfuscation (b) Obscuring trial versions (c) Integrating obfuscator to IDE

Also, any trial versions, in Figure 4(b), of the freeware or a product can be obscured and uploaded to the server or supplied to the client, to curtail down possible attacks.

We also suggest, in Figure 4(c), to integrate the obfuscator component to the existing Java Integrated Development Environment(IDE) like Eclipse, to facilitate obfuscation during code writing, by helping the coder to include built in CoBS classes and to utilize its member methods.

## 4. Conclusion

We claim that the utilization of obfuscation tools, to an extent, would protect codes from intellectual property violation attacks. Hence, we recommend using the obfuscator on the product codes, before releasing it to any third party. Industry can develop their own repository of classes and can enhance their repository, for product development, to implement better security.

## 5. References


[1] C. Collberg,C. Thomborson,D. Low. A Taxonomy of obfuscating Transformations. Report 148. Department of Computer Science. University of Auckland, 1997.

[2] Collberg,C. Thomborson. Watermarking, Tamper-proofing and obfuscation – tools for software protection IEEE Transactions on Software Engineering, Vol. 28,pp. 735-746, 2002.

[3] S. Praveen,P. Sojan Lal,S. Naveen. JDATATRANS for Array Obfuscation in Java Source Codes to Defeat Reverse Engineering from Decompiled Codes. Proc. of 2nd Bangalore Annual ACM COMPUTE Conference, ACM Digital Library,2009.

[4] E.E.Mills. Software Metrics. Technical Report. Software Engineering Institute, Carnegie Mellon University,1988.

[5] M. Madou, B. Anckaert, D. Bus, D. Bosschere, J. Cappaert, B. Preneel. On the Effectiveness of Source Code Transformations for Binary Obfuscation. Proc. of the International Conference on Software Engineering Research and Practice (SERP06), 2006.

[6] S. Praveen, P. Sojan Lal, S. Naveen. JConstHide: A Framework for Java Source Code Constant Hiding. Journal of Information Assurance and Security (JIAS), 4(1), 2009.

[7] A.Majumdar. Design and Evaluations of Software Obfuscations. PhD Thesis. University of Auckland, Newzealand, 2008.

[8] W.F. Zhu. Concepts and Techniques in Software Watermarking and Obfuscation. PhD Thesis. University of Auckland, Newzealand,2007.

[9] H.A.Muller, J.H.Jahnke, D.B.Smith, M.A. Storey, S.R.Tilley, K.Wong. Reverse Engineering: A Roadmap. ACM 2000.

[10] C.Collberg, C.Thomborson, D.Low. Breaking Abstractions and Unstructuring Data Structures. Proc. of IEEE International Conference on Computer Languages (ICCL '98), 1998.

[11] F. M. Kugblenu. Reverse Engineering – Tools and techniques for obfuscating Java Byte code. Technical Report. Blekinge Institute of Technology.

[12] G. Nolan. Decompiling Java. Apress, First Edition, 2004.