IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-3, Issue-5 E-ISSN: 2347-2693

A Thorough Investigation of Code Obfuscation Techniques for Software Protection

Krishan Kumar[1], Prabhpreet Kaur[2]
[1*,2]Department of Computer Science, Guru Nanak Dev University, INDIA
www.ijcseonline.org

Received: Apr/22/2015 Revised: May/01//2015 Accepted: May/20/2015 Published: May/30/ 2015

Abstract: The Process of reverse engineering allows attackers to understand the behavior of software and extract the proprietary algorithms and key data structures (e.g. cryptographic keys) from it. Code obfuscation is the technique is employed to protect the software from the risk of reverse engineering i.e. to protect software against analysis and unwanted modification. Program obfuscation makes code harder to analyze. In this paper we survey the literature on code obfuscation. we have analyze the different obfuscation techniques in relation to protection of intellectual property. At the last, we are purposing suggestion to provide protection from both the static and dynamic attacks.

Keywords— *Code Obfuscation, Software Protection, Reverse engineering*

1. Introduction

In the last decade, code of the software is distributed in an architecturally-neutral format which has increased the ability to reverse engineer source code from the executables. This activity has greatly concerned by the software companies who has desire to protect the intellectual property of their products. As there are lots of copyright laws which forbid the direct piracy of software, most of the developers are worried by possible theft of proprietary data structure and algorithms design. Though there are several methods for protecting software, such as encryption, server-side execution and native code, obfuscation has been found to be the cheapest and easiest solution to this problem [1]. So main target of code obfuscation is to protect the sensitive information such as data structure and algorithms of a software from getting disclosed to the outer world and only technique which is available in digital market to get the sensitive information's about the proprietary or intellectual properties from the executable revere engineering. Code obfuscation is the only technique that can prevent reverse engineering to some extent to analyze the target software.

Code obfuscation is the practice of making code unintelligible, or at very least, hard to understand. The process of code obfuscation involves transforming the code of application to the code which is difficult to understand by changing the physical appearance of the code, while preserving the black-box specification of the program. Obfuscation, by being the transformation of the program, can be understood as the special case of data coding. The further analysis shows, that there are a lot of similarities between obfuscation and cryptography, but still these two techniques cannot treated as equivalent. In this paper we have surveyed the different obfuscation techniques [2].

Code obfuscation not only used by developers to protect intellectual property, it is also used extensively by malware writers to avoid detection. Many viruses utilize obfuscation techniques to subvert virus scanners by continually changing their code signature with obfuscating transformations.

2. Code Obfuscation

Code obfuscation technique is to obscure the control, data, layout, design of the software original implementation and give a semantically same but new implementation.

There is no common formal definition for code obfuscation. It is basically transformation method to convert one program into another, which posses the same characteristics of the old program. It can also be treated as an executables that contain encrypted sections, and a simple code section to decrypt the encrypted code section. According to the authors of the paper "A taxonomy of obfuscating transformations" [3], the definition of code obfuscation is as follows:

Definition: Let T(P) be a transformed program of program P. Then T is the Obfuscating Technique if T(P) poses the same observable characteristics as P and T(P) must follows the following conditions:

If program P does not terminate or has an erroneous termination, then T(P) may or may not terminate. Else as P terminates successfully, T(P) must terminate with the same outcome as P.

According to the authors of the paper "A security architecture for survivability mechanisms" [4], if T is obfuscating technique that transform the program P into the obfuscated binary B, then the reverse transformation from B to P will take much greater effort and time(almost impossible),as T is a one way translator.

2.1 Classification of code obfuscation

Obfuscation is classified into four types [3] based on obfuscation target - Layout obfuscation, Data obfuscation, Control obfuscation and Preventive transformation.

2.1.1. Layout Obfuscation: It refers to obscuring of the software layout by deleting comments for instance, changing format of the source code, variables renaming, and the removal debugging information through obscuring the lexical structure of the program.

2.1.2 Data Obfuscation: This prevents the extraction of information from data. Data obfuscation techniques are array splitting, variable splitting, changing the scope and lifetime of data etc.

2.1.3 Control Obfuscation: it refers to the obscuring of the control flow of the program. This kind of obfuscation technique mainly of dynamic obfuscation type based on self modifying code.

2.1.4.Preventive Transformation: Depending on debuggers' or disassemblers' weaknesses, modify the program such that code itself will force the debugger or disassembler to fail.

But this classification does not include all types of obfuscation techniques. Another possible classification is Design Obfuscation [5] which deals with obscuring the design related information's of the software. Like merging and splitting of code sections or classes, type hiding, will help in obscuring the design intend of the programs.

3. Different evaluation criteria's to measure the effectiveness of Code Obfuscation

The three sets of Criteria (A-1, A-2, A-3) are described in the subsections below and another method called empirical is also described.

3.1 A-1: Potency, Resilience, Cost(Analytical Methods)

Analytical method checks the quality of the obfuscating technique T() depending upon the parameters of both original/source program P and the obfuscated program T(P). According to authors of the paper "A taxonomy of obfuscating transformation" [21], they are evaluating the quality depending upon three parameters - potency, resilience and cost.

Potency: It can be described as - how much obscurity T() adds to P. Let Pot(P) is the potency measurement of P and Pot(T(P)) is the potency measurement of T(P) then Transformation Potency,

$$TPot=Pot(T(P))/Pot(P)-1 \quad (3.1)$$

Cost: It is measure by how much computational overhead T() adds to T(P). It is the execution time penalty and space penalty that the obfuscation technique incurs on T(P). If executing T(P).
requires exponentially more resources than P then Transformation Cost;

$$TCost = Dear \quad (3.2)$$

if executing T(P) requires $O(n^p)$, p>1, more resources than P then Transformation Cost;

$$TCost = Costly \quad (3.3)$$

If executing T(P) requires O(n), more resources than P then Transformation Cost;

$$TCost = Cheap \quad (3.4)$$

If executing T(P) requires O(1), more resources than P then Transformation Cost;

$$TCost = Free \quad (3.5)$$

Resilience: It is measured by how difficult is T(P) to break for a deobfuscator means how well a T() holds up under attack from a automatic deobfuscator. Resilience can be measured by summing the total of programmer's effort and deobfuscator's efforts [3].

Programmer Effort (PEff) - The amount of time require by the programmer to build the automatic deobfuscator to regenerate P from T(P).

Deobfuscator Effort (DeoEff) - The amount of execution time and space required for the automated deobfuscator to deobfuscate the transformed program.
If P cannot be constructed from T(P), means some information from P is removed in T(P) at the time of obfuscation, then Transformation Resilience;

$$TRes = One\ Way \quad (3.6)$$

Otherwise

$$TRes = Res\ (PEff + DeoEff) \quad (3.7)$$

3.2 A-2: Resistance to Static and Dynamic Attacks

Madou and et. al [20] describe static and dynamic attacks carried out on software. Static attacks are based on static information and it is obtained by examining and analyzing program without executing it. Dynamic attacks are solely based on dynamic information. It is obtained by executing program and observing execution traces. They measure effectiveness of code obfuscation based on resistance of obfuscated code to static and dynamic attacks. Sebastian and others [21] also describe effectiveness of code obfuscation based on resistance to static and dynamic reverse engineering attacks.

3.3 A-3: Increase in Program Static Space

Chow and et. al in [22] describe obfuscation of control flow of program by expanding state space of program.

This is achieved by embedding an instance "I" of hard combinatorial problem "C" into code of program. It is necessary to find the solution ("K") to the instance (by static analysis) which is needed to detect essential property "P" of code. This obfuscation technique expands state space of program (called dispatcher code) and this paper shows that it is not possible to minimize its state space. Thus, if state space of obfuscated program is larger than original program, reverse engineering efforts by attacker are increased. Hence

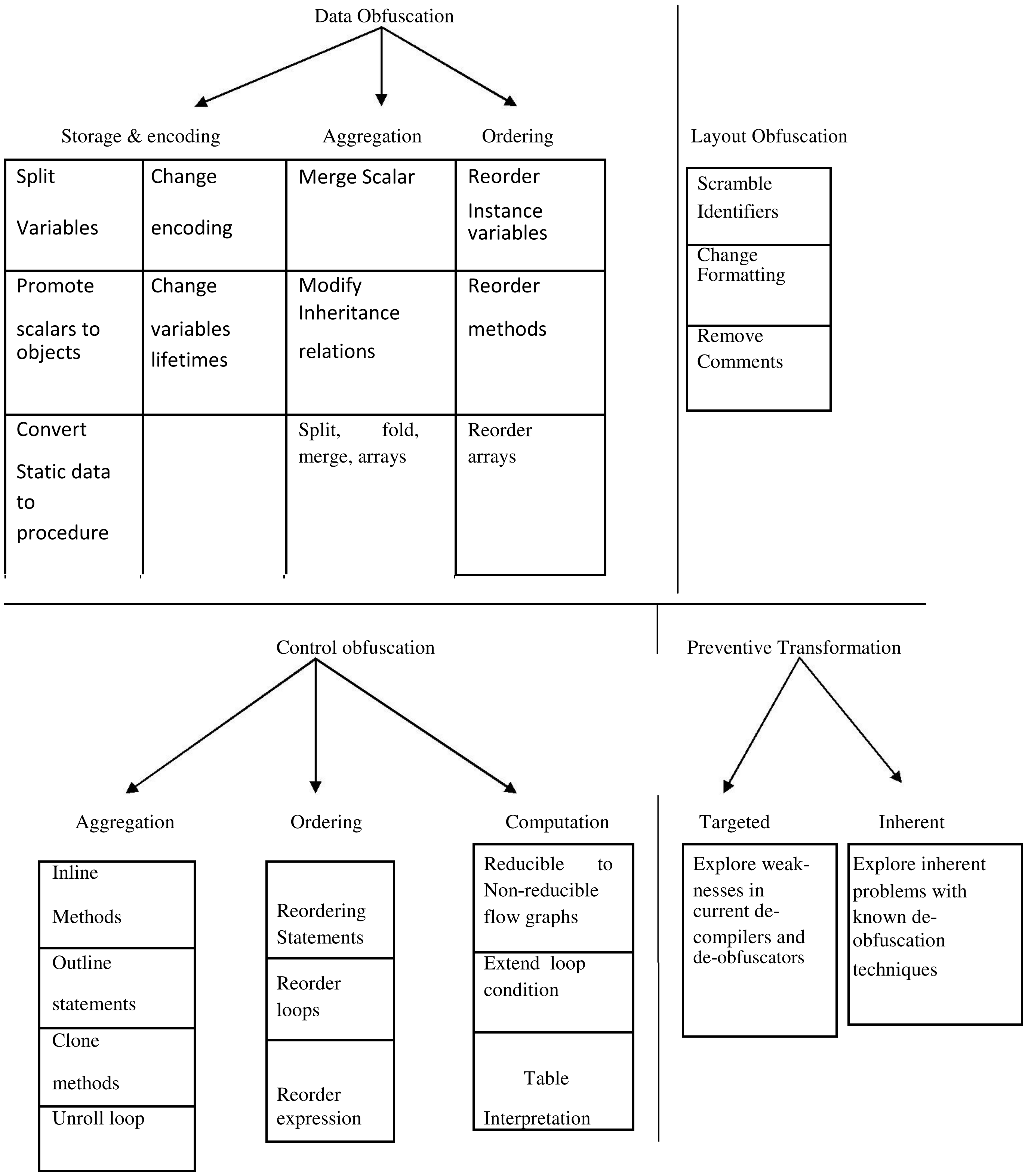

Figure 1: Classification of obfuscating transformation

such code obfuscation technique is better that technique which does not increase state space of program.

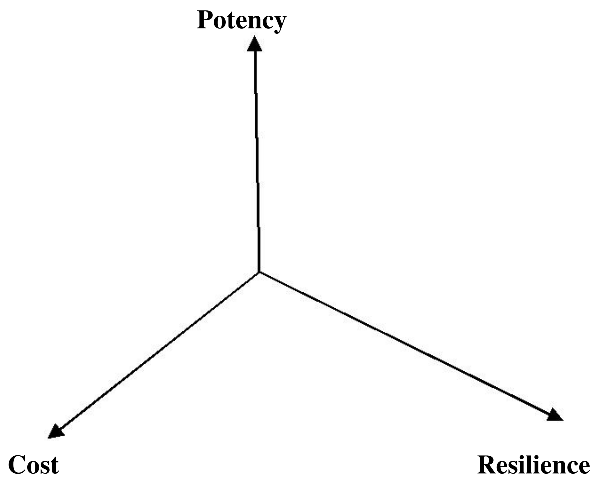

Figure 2: Metrics for quality measurement of the obfuscation technique

3.4 Empirical Method

The main target of code obfuscation is to protect the proprietary code sections or algorithms from unauthorized analysis and in reverse engineering the last step of analysis is totally depends on human effort which cannot be measured by any metrics. For this we need to perform empirical research on a group of people like programmers, hackers or crackers, students [9].

4. Literature Surveyed

In this section we have studied different obfuscation techniques which appeared in literature and drawback of each technique is also discussed.

Software protection through dynamic code mutation – 2006

The researchers of this paper [11] implement a dynamic code obfuscation technique that will remove some set of code which will be restored at run time. To implement this idea they are using three extra code module - stub, edit script and edit engine. First thing they are doing is the identification of basic blocks, then they are removing a set of code from a basic block and put the restoring informations in a edit script. Afterwards they include a stub, which will have the address of the corresponding edit script, at the beginning of that block and desperately put some confusing erroneous code on place of removed set of code. At the time of execution stub will be executed first and transfer the control to edit engine with the address of corresponding edit script. Then according to edit script the edit engine will restore the original set of code at position of the erroneous set of code. This method is implemented in two ways by the researchers of this paper. One is One-Pass Mutation where each functions or basic blocks will have their own edit script. Other one is Cluster-Based Mutation where a group of similar functions will have a single edit script. The major disadvantage of this technique is the stub section is always be in highlight, that will draw attention of the attacker. Other disadvantage is after restoring, the original code is fully exposed to the debugger or attacker.

Binary obfuscation using signals - 2007

Here [12] the researchers also give a new technique of control flow obfuscation by hiding the control flow information of a program using signal, which are used carry information between operating system and information. This research work is based on the replacement of every control instruction at binary level (eg. JMP,RET, CALL) with trap signals like SIGILL forlegalinstruction, SIGSEGV for segmentation violation and SIGFPE for floating point exception. It first identify the control instruction, then divides the code-before and code-after segment of the control instruction. After this the control instruction is replaced with a trap instruction and some bogus code is inserted between the trap instruction and the code-after segment. Then the user defined signal handlers are installed within the program with a special table that will contains the actual instruction for corresponding generated signals. At runtime when the trap signal will executed the control will go to the operating system's corresponding signal handlers, then the control will be transferred to user-defined signal handler for the corresponding signal. Then the user-define signal handler will execute the corresponding code and then transfer the control to the code-after segment. The one disadvantage of this technique is the control instruction is available within the user-defined signal handler. If the attacker can identify the signal handler, he can identify the control instruction by analyzing the signal handler.

Mimimorphism: a new approach to binary code obfuscation - 2010

In this paper [13] the authors give a totally different kind of obfuscation technique based on mimic function that has three phases - a digesting phase for Huffman tree building, an encoding phase that use Huffman decoding technique and a decoding phase that use Huffman encoding technique. Here the mimimorphism technique use mimic function of higher order which differ in digesting phase from regular mimic function by building a collection of Huffman trees for better mimicry and a mimimorphic engine, that include all the three phases, is added to the obfuscated program to restore the original code at run time. Here, in Digesting phase, from the executable with help of an assembler for each assembly instruction with all the parameters and the frequency of occurrence those parameters are stored and all the instruction is also get stored with a unique id and with the frequency of their occurrence, after this a Huffman tree for each instruction is created depending on

their parameters frequency. At encoding phase this technique use the Huffman decoding operation base on the Huffman trees
generated earlier in digesting phase and output a completely different assembly code that will convert into a binary with the help of an assembler. At execution time of the new binary code the mimimorphic engine apples its decoding function on the binary, that use the Huffman encoding operation, depending on the Huffman trees generate earlier to restore the original program for execution.

Here the binary code that will be distributed can't be reverse engineer statically but it includes the mimimorphic engine, the decoder with the Huffman trees with unobfuscated status. This may reveal the original code with dynamic analysis and also encoding and decoding the whole program is very time consuming when program size will increase.

Mobile agent protection with self-modifying code – 2011

The paper [14] introduces a light weight but self-modifying code based technique at binary level. The proposed obfuscation technique of the this paper camouflaged the control instructions with normal instructions or with other control instructions. This method defines each control instruction as a candidate block, the code section before the candidate block is named as preceding block and the code section afterwards is as succeeding block. At the time of obfuscation this technique replace the control instruction (for example JMP instruction) at the candidate block with normal instruction (for example MOV instruction) and add a modifying block to its preceding block and add a restoring block to its succeeding block.

The modifying block performs some AND-OR operations on the address of the candidate block to restore the original instruction at run time. After execution of the candidate block when control goes to the succeeding block, then restoring block again perform some AND-OR operations the address of the candidate block and restore the camouflaged instruction again in the candidate block at runtime.

The obfuscated code developed by this method will not be too much bigger than the original one, as no extra code section is add, instead 2-4 simple binary level code is added to the original binary one. This kind of obfuscation is very hard to be found by static reverse engineering and make the analysis error prone. But the original is exposed temporarily at the time of execution which can be detected by dynamic reverse engineering with the help of any debugger [18] and also the modifying and restoring block can be identified by step-in execution(execute one instruction at a time) within the debugger.

Branch obfuscation using code mobility and signal – 2012

The research work [15] provides a obfuscation technique where resilience [3] is one-way means the original program cannot be reconstructed from the obfuscated one. On the basis of the paper "Binary obfuscation using signals" [12] the researchers of this paper build their work. They are also using the trap instruction in place of the control instruction, that they want to be obfuscated. In the same way of the base paper [12] they removed the control instruction and put a trap instruction with bogus codes afterwards. When the trap instruction will execute depending on the generated signal control will transfer to operating system, then to the corresponding installed user-define signal handler. Here the signal handler will communicate to a remote trusted server/machine by passing the value of the actual condition variable to know the next code section that will going to be executed next. On receiving the value of the condition variable the server generate the corresponding result and pass it to the signal handler, which will then pass the control to the next executing block depending on value of the result. Here they are not providing the complete executable code to the customer. They are removing some information from the provided binary one and add server-side execution of the removed information, code obfuscation technique is only used to hide the actual control instruction form the attacker.
This a hybrid method of code obfuscation and server-side execution. As some code is removed from the provided binary, the original code can never be reconstructed from the binary with the help of any kind of reverse engineering. But the performance of this code totally depends upon the connectivity of between the two machines. If the network bandwidth is too low or there is no connectivity between the two machines, this implementation is totally worthless.

Potent and stealthy control flow obfuscation by stack based self-modifying code – 2013

Here [16] the researchers developed a stronger new obfuscation technique based on the paper "Mobile agent protection with self-modifying code" [14] described earlier. On the previous paper they are just trying to hide the control instruction but the address where the control will be transferred is still available after camouflaged. Here the researchers have shown a way to hide the address also as a local data to that function, which will be stored on the stack section of data area. In this research work the researchers take executable machine code and then generate its corresponding assembly code. Then they select the control instruction to be obfuscated. Lets take they are going to obfuscate a JMP instruction(an assembly instruction for unconditional jump with a address parameter). So to store the address in the stack they are just extending the size of stack that will always be allocated at the starting point of the function. After this before obfuscating the instruction they stored the jump address in the stack and then replaced the JMP instruction with a normal instruction and add an extra instruction in the modifying block after the deobfuscation instructions to restore the address at run time and an extra instruction to restoring block to remove the address at run

time, before the re-obfuscation instructions.
This method provide a code obfuscation mechanism that is to hard to be analyzed by static reverse engineering as both address and the instruction is not visible until the function stores its stack onto the memory. This thing also make it hard for dynamic reverse engineering. But in modern debuggers [18] [17] [19] if we execute the obfuscated binary with the step-in (execute one instruction at a time) execution it will shows all possible values of every registers and stack pointer, local and global variable values used at that moment.

Dynamic Obfuscation Algorithm based on Demand-Driven Symbolic Execution -2014

In this [22] author has presented a novel algorithm called Demand-Driven Dynamic(DDD) Obfuscation by using the demand driven theory of symbolic analysis. In this algorithm, first large number of invalid paths are created that mislead the result of symbolic analysis. Secondly, according to this theory, a specific execution path is created to protect the security of software. The DDD algorithm proposed four important obfuscation concept: jump node(*jn*),node summery, target driven and program components. Main component of the algorithm is jump node. Program execution path can be changed and controlled by inserting *jn* and logic of *jn*. Each *jn* correspond to a unique ID which is used to locate the node summery information in the lookup table through hash function and then to determine the relative position of the *jn* in the execution tree.

In this algorithm the efficiency and performance depends on jn. The position and number which *jn* added need to consider efficiency and safety performance after confusion. When number of *jn* is large, it may reduce the program execution performance, so that the number and density of *jn* need to be controlled.

An Obfuscation Method to Build a Fake Call Flow Graph by Hooking Method Calls-2014

The researcher in the paper [23] proposes an obfuscation method against the illegal analysis of program code. This method tries to build a fake call flow graph with help of debugging tools. The generated call flow graph shows relations among methods, and helps in comprehension of a program. The main concept is that call flow graph leads to misunderstanding of the program. It is implemented though hook mechanism of the method call from changing a callee. The key idea of the proposed method is to change a callee before runtime, then the actual callee is called by the hook method at runtime. In the early stages of attack, the adversaries try exposing that what kind of protection mechanism is used. If the protection method is stealthy, the attacks for the program become more hard task.

Here the researcher uses the invokedynamic instruction and classes in java.lang.invoke, general java program rarely used them. This is the place where attacker can easy identify the method. The proposed method provides protection against the static analysis but it fails against the dynamic analysis.

5. Conclusions

As the goal of code obfuscation is to protect the sensitive information such as data structure and algorithms of software from getting disclosed to the outer world and it can thwart many attacks but with enough time and efforts above discussed techniques can be overcome by reverse engineers. Researchers have found many code obfuscation techniques but no obfuscation technique has yet been found that can completely resist reverse engineering. In addition to this drawback, code obfuscation increases the code foot-print, decreases little bit performance, and can hinder certain compiler optimizations. When obfuscation techniques combined appropriately, can add a layer of protection against illegal modifications, theft and insertion of malicious code. The literature surveyed many obfuscation techniques each having some limitations some provide protection against static reverse engineering other against dynamic reverse engineering. So there is need for Hybrid obfuscation mechanism which provide protection from both the static and dynamic reverse engineering. So our future work will be about developing Hybrid mechanism for obfuscation. Several software protection techniques available in the literature are analyzed and examined. The characteristic features of the existing algorithms are thoroughly investigated in this paper. This study would facilitate in development of efficient software protection techniques. Encryption techniques can be incorporated with the existing software protection techniques to improve the overall security of the software. Code encryption schemes for protecting software against various attacks like reverse engineering and modification. Therefore, novel and efficient code encryption scheme have to be established based on an indexed table to guarantee secure key management and efficiency.

References

[1]Christian S. Collberg and Clark Thombor-son. Watermarking, tamper-proofing, and obfuscation - tools for software protection.In IEEE Transactions on Software Engineering, volume 28, pages 735–746, August 2002.

[2] Shakya Sundar Das, Code Obfuscation using Code Splitting with Self-modifying Code, Disseration National Institute of Technology Rourkela -769 008, Odisha, India May 2014.

[3] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," tech. rep., Department of Computer Science, The University of Auckland, New Zealand, 1997.

[4] C. Wang, A security architecture for survivability mechanisms. PhD thesis, University of Virginia, 2001.

[5] V. Balachandran and S. Emmanuel, "Potent and stealthy control flow obfuscation by stack based self-modifying code," Information Forensics and Security, IEEE Transactions on, vol. 8, no. 4, pp. 669-681, 2013.

[6] M. H. Halstead, Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., 1977.

[7] T. J. McCabe, "A complexity measure," Software Engineering, IEEE Transactions on, no. 4, pp. 308-320, 1976.

[8] W. A. Harrison and K. I. Magel, "A complexity measure based on nesting level," ACM Sigplan Notices, vol. 16, no. 3, pp. 63-74, 1981.

[9] G. Wroblewski, General Method of Program Code Obfuscation (draft). PhD thesis, Citeseer, 2002.

[10] C. Wang, A security architecture for survivability mechanisms. PhD thesis, University of Virginia, 2001

[11] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, and K. De Bosschere, "Software protection through dynamic code mutation," in Information Security Applications, pp. 194-206, Springer, 2006.

[12] I. V. Popov, S. K. Debray, and G. R. Andrews, "Binary obfuscation using signals," in USENIX Security Symposium, pp. 275-290, 2007

[13] Z. Wu, S. Gianvecchio, M. Xie, and H. Wang, "Mimimorphism: a new approach to binary code obfuscation," in Proceedings of the 17th ACM conference on Computer and communications security, pp. 536-546, ACM, 2010.

[14] L. Shan and S. Emmanuel, "Mobile agent protection with self-modifying code," Journal of Signal Processing Systems, vol. 65, no. 1, pp. 105-116,2011

[15] Z. Wang, C. Jia, M. Liu, and X. Yu, "Branch obfuscation using code mobility and signal," in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, pp. 553-558, IEEE, 2012.

[16] V. Balachandran and S. Emmanuel, "Potent and stealthy control ow obfuscation by stack based self-modifying code," Information Forensics and Security, IEEE Transactions on, vol. 8, no. 4, pp. 669-681, 2013.

[17]"Idapro debugger : Data rescue [Online]." http://www.datarescue.com/. Last Accessed: 07-1-2015

[18]"Immunity debugger [Online]."
https://www.immunityinc.com/productsimmdbg.shtml. Last Accessed: 07-1-2015.

[19] "Olly debugger [Online]." http://www.ollydbg.de". Last Accessed: 12-09-2013

[20] Matias Madou, Bertrand Anckaert, Bjorn De Sutter, and De Bosschere Koen."Hybrid static-dynamic attacks against software protection mechanisms", In Proceedings of the 5th ACM Workshop on Digital Rights Management. ACM, 2005

[21] Sebastian Schrittwieser and Stefan Katzenbeisser, "Code Obfuscation against Static and Dynamic Reverse Engineering", Vienna University of Technology, Austria, Darmstadt University of Technology, Germany.

[22] Chow, S., Gu, Y., Johnson, H., and Zakharov, V.A.: "An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs", In the proceedings of 4th International Conference on Information Security, LNCS Volume 2200. Pages 144-155. Springer-Verlag. Malaga, Spain. 2001.

[23]F.Kazumasa , T.Haruaki " An Obfuscation Method to Build a Fake Call Flow Graph by Hooking Method Calls" Las Vegas, USA IEEE, SNPD 2014, June 30-July 2, 2014.