Journal of Theoretical and Applied Information Technology
15th December 2012. Vol. 46 No.1
© 2005 - 2012 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A CONTROL FLOW OBFUSCATION SCHEME BASED ON GARBAGE CODE

[1,2]YONGYONG SUN, [1]GUANGQIU HUANG

[1]School of Management, Xi'an University of Architecture and Technology, Xi'an 710055, Shaanxi, China

[2]School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710032, Shaanxi, China

ABSTRACT

Control flow obfuscation is used to obfuscate program execution flow, thus preventing reverse engineering of software. However, the code size and execution time will be increased greatly after program is obfuscated. Since opaque predicate is difficult to resist the dynamic attack, the paper proposes the scheme about control flow obfuscation based on garbage code. The algorithm combines branch garbage code with loop garbage code, and hash function is introduced to limit the number of insert operation about code. Thereby controlling the growth of the code size and reducing the accuracy of program analysis and resisting tampering attack. Experimental results show that the obfuscation algorithm can control performance overloading effectively that is brought by obfuscation transformation, while preventing a variety of reverse engineering attacks effectively.

Keywords: C*ontrol Flow Obfuscation, Reverse Engineering, Opaque Predicate, Garbage Code, Hash Function*

1. INTRODUCTION

At present, more and more software is released as a platform-independent intermediate code. The code is easier to attack maliciously than the traditional binary executable code, such as static analysis, reverse engineering and tampering [1]. With the development of network technology, a lot of software will be run under an uncertain environment, and the software can be analyzed and tracked randomly by host. Moreover, with the development and widespread application of various reverse engineering technology, the attack on software becomes easier [2]. How to protect the core algorithms and confidential data in the software is a focus. Technology of code obfuscation is an important protection method for software [3]. It means that transforms program by maintaining the semantics, so that the program after transformation has the same or similar function with original program, but more difficult to attack by static analysis and reverse engineering [4, 5]. The purpose of code obfuscation is not to provide absolute protection for software, and it makes the attackers abandon attack behavior because the cost of attack is far more than they can afford.

Many scholars have proposed the scheme on Java byte code obfuscation. Code security system against malicious host is established, and its obfuscation algorithm core is to destroy the program control information, then, undermine the call information in process of program. For the characteristics of Java language, the difficulty of understanding on the code is increased by constructing complex data structure, control flow information in process is obfuscated by inserting the multi-branch statements [6, 7]. We can know that the research on program data flow is less than program control flow. Many obfuscation algorithms of control flow use opaque predicate or the design gap between Java language and JVM to obfuscate program. These obfuscation methods are difficult to resist the dynamic attack and de-obfuscation attack of pattern matching [8, 9].

In this paper, the algorithm of control flow obfuscation based on garbage code is proposed, and the hash function is introduced to limit insert operation of code, thereby controlling the growth of code size. The algorithm of control flow obfuscation based on Java byte code is designed and implemented, and the algorithm combines branch garbage code with loop garbage code. It can implement the iteration obfuscation of Java byte code. Moreover, the obfuscation results are irreproducible.

2. DEFINITION OF CODE OBFUSCATION

Code obfuscation essence is to provide a translation mechanism [10], and the program after transformation is difficult to be understood.

Obfuscation transformation can be divided into five types, which is layout obfuscation, control flow obfuscation, data obfuscation, class structure obfuscation and preventive obfuscation. The definition of obfuscation transformation can be described as definition one.

Definition one: The program $P(v_1)=v_2$ is changed into the program $\tau(P)$, then, there is a program *P*', original output context of program *P* is recovered, and $\tau(P)|P'$ is equivalent to the program *P* when the context *S* is inputted and the context *S*' is output. That is called obfuscation transformation. $\tau(P)(v)=v_\tau$ is the obfuscation transformation of $P(v)=v'$ which is equivalent to $(\underset{P'}{\vee} P'(v_\tau)=v' \wedge \tau(P)|P' \equiv P(v)=v')$.

Instruction sequence *P*' is called the end of obfuscation transformation. Some instructions can not change the context, such as input instruction, output instruction, so not all instructions given architecture can be obfuscated transformation. The obfuscated program is divided into *τ*(*P*) and *P*', it is possible to bind reversible algorithm for obfuscation transformation. Equivalence conditions $\tau(P)|P' \equiv P(v)=v'$ means program $\tau(P)(v)=v_\tau$ and $P(v)=v'$ is constant mapping.

Branch garbage code and loop garbage code are combined in this paper, including the determination of basic program modules and opaque predicate. The basic modules and opaque predicate are defined as definition two and definition three.

Definition two: Program basic block is an instruction sequence. There is not any transfer instruction in the sequence. The range of module address does not contain not only any transfer instruction but also the target address of transfer instruction.

Definition three: The value of predicate *P* is obfuscated when it is *p*, it can be gotten by obfuscator not de-obfuscator,, then, the predicate is opaque. It is described that $P_p^F(P_p^T)$ always is *False* or *True* when its value is *p*. If its value sometimes is *True* and sometimes is *False*, it is defined as $P_p^?$.

3. THE REALIZATION ON OBFUSCATION ALGORITHM

There is not any code to meet the condition of the garbage code insertion. It is not easy to construct garbage code, and it is more difficult to propose a universal practical garbage code algorithm. Branch garbage code and loop garbage code are described in this paper.

3.1 The Definition Of Garbage Code

Garbage code is the code that is inserted into the Java byte code array, and it can be executed during the Java program is running. But it does not change the operation results. Garbage code has the characteristics as the following.

(1) Whether any class or instance variable in the Java program is referred, its value can not be changed. But the value of the class or instance variable that is added by obfuscator can be changed.

(2) The value of any local variable in the original Java program can be referred, while the value can not be changed.

(3) Assuming that the state of the stack is *A* before the control flow comes into the garbage code, no matter what the place where control flow left from the garbage code, the state of stack must maintain *A*.

(4) Garbage code can create objects in heap, but the state of original objects in heap can not be changed.

(5) Garbage code does not change the original control flow of Java method. That is, if the original control flow is from *A* to *B* before garbage code is inserted, the control flow is from *A* to *C* after garbage code *C* is inserted, and the control flow must turn to *B* after the garbage code *C* is run.

The above characteristics can ensure the operational environment of original program instruction not to be changed. At the same time, control flow of the original program can not be changed. So the running results of Java method also can not be changed.

3.2 Branch Garbage Code

The process of obfuscation transformation about branch garbage code is shown in Figure.1.The realization on algorithm is described as following.

Input: Given the program control flow graph *G*.

Step1. Examine all basic blocks in control flow graph *G*, find some basic blocks *B*, and the state of blocks *B* are no change when program control flow leaves them, and data of stack top is numeric. Insert an opaque predicate $P_p^F(P_p^T)$ to interfere the statements execution after control flow.

Step2. Add a new local variable in Java byte code, and its type is double. Add pseudo instruction sequence *insert id* and *store index* before the basic blocks that is selected in Step1, *id* is unique identification of a basic block, *insert* is any instruction that pushes float data into stack, and

store is any instruction that stores float data in local variable.
Step3. Introduce a class variable, and the type of variable is the same with the type of stack top element that control flow left any basic block in *B*.
Step4. Construct the following pseudo instruction sequence: *open*, *put static index _ pool*, *load index*, *lookup switch*. The jump table of *lookup switch* is constructed as the following way: *id* and a certain subsequence in basic block that is corresponding with *id* turn to *open*, *put static index _ pool*, *load index*, *lookup switch*, not turn to branch of the subsequence.
Step5. Instruction sequence that is constructed in Step4 inserts the basic block that is selected in Step1. At the same time, the branch of basic block points to *open*, *put static index _ pool*, *load index*, *lookup switch*.
Output: Program control flow graph *G*'.

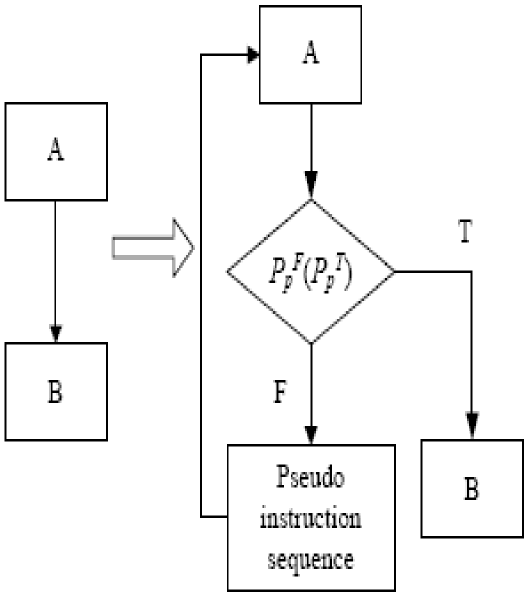

Figure 1: Obfuscation Transformation Of Branch Garbage Code

3.3 Loop Garbage Code

The local control flow after the loop garbage code is inserted is shown in Figure.2. The realization on algorithm is described as following.

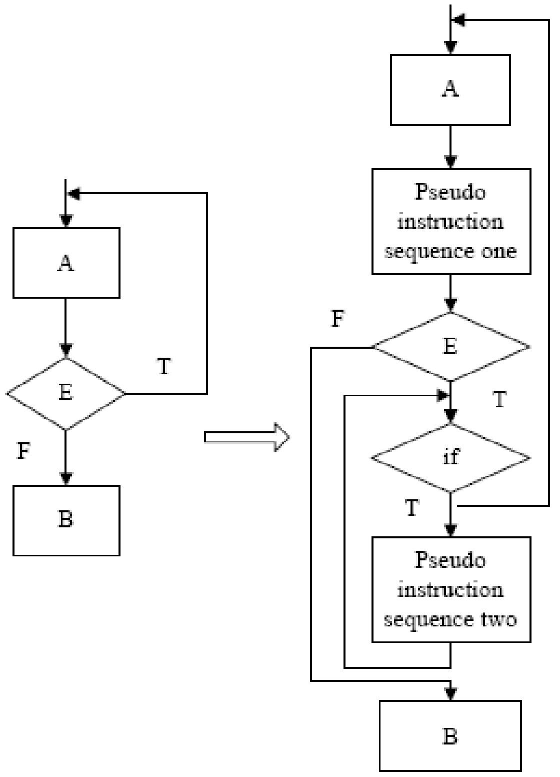
A
E
T
F
B
A
Pseudo instruction sequence one
F
E
T
if
T
Pseudo instruction sequence two
B

Figure 2: Obfuscation Transformation Of Loop Garbage Code

Input: Given the program control flow graph *G*.
Step1. Examine control flow graph of Java byte code and find basic block *B*. The basic block should have several precursors. And the stack top element is numeric before control flow came into the basic block.
Step2. Add a new local variable in Java byte code, and its type is double.
Step3. Introduce a class variable, and the type of variable is the same with the type of stack top element that control flow came into basic block *B*.
Step4. Construct the following pseudo instruction sequence one: *insert id*, *open*, *store index*, *if*, *insert pool*, *load index*, *add*, *open*, *store index*, *go to*. And *insert* is any instruction that pushes float data into stack, *store* is any instruction that stores float data in local variable, *if* is conditional branch instruction, its target is basic block *B*, *load* is any instruction that loaded *integral* data from local variable into stack, *go to* is unconditional transfer instruction, and its transfer target is *if* instruction of the new code, *pool* is any float constant.
Step5. Instruction sequence that is constructed in Step4 is inserted before the basic block that is selected in Step1. At the same time, a certain original precursor of the basic block points to the first new instruction, and other precursors point to

the new pseudo instruction sequence two: *insert pool*, *load index*, *add*, *open*, *store index*, *go to*, *if* instruction is the only exit for instruction sequence *insert id*, *dup*, *store index*, *if*, *push const*, *load index*, *add*, *dup*, *store index*, *go to*.
Output: Program control flow graph *G'*.

3.4 The Insertion of Hash Function

The obfuscated program will bring the cost of execution time and required storage space. Even it can not run because the memory space is run out. So the obfuscation method should be improved and limit the number of obfuscation operation. In this paper, the improved idea is to use hash function to select operation modules to obfuscate when operation modules are too large, in order to limit the number of obfuscation operation and reduce the cost of time and space on obfuscated program.

4. PERFORMANCE ANALYSIS OF THE ALGORITHM

The effectiveness and feasibility of obfuscation algorithm about control flow based on garbage code is verified by CPU clock cycle and the success rate of the implementation. Since the control flow of general Java program is mainly made of several control statements, six different Java programs are selected for the experiment. Test case is shown in Table 1. The capability of resist tampering on garbage code is compared with the algorithm of class structure. The experimental results are shown in Figure.3 and Figure.4.

Table 1 : Test case

Number of program	Name of program	Description
1	P_if	if-else program
2	P_for	for program
3	P_while	while program
4	P_do-while	do-while program
5	P_Switch	switch program
6	P_ colligate	compositive program

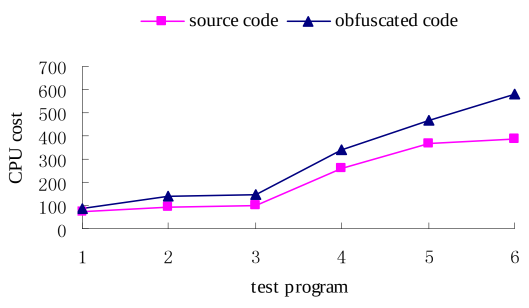

Figure.3: Execution Cost Of Source Code And Obfuscated Code

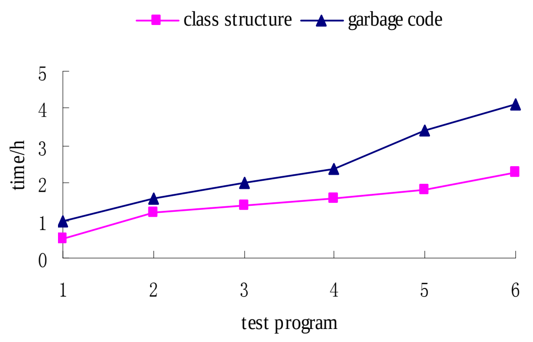

Figure.4: Comparison Of Ability On Resisting Tampering

Through the experimental results, it can be seen the execution cost of obfuscated program is larger than the original program, no matter what the original code size. In addition, with the growth of program complexity, the capability of resist tampering about the obfuscation algorithm based on garbage code is better than the obfuscation algorithm of class structure. The algorithm can extend the time of tampering attack, and prevent static analysis and reverse engineering for software.

5. CONCLUSION

In this paper, garbage code is used to implement obfuscation algorithm of control flow, and the realization processes about branch garbage code and loop garbage code are provided. Furthermore, the number of insertion operation about garbage code is limited by hash function. It can ensure the code obfuscation strength to some extent, and it also can avoid code not to run properly because of excessive cost. The capability on resisting dynamic attack of garbage code is stronger than the opaque predicate, and the garbage code also can prevent static analysis and reverse engineering for software effectively. The obfuscated program has a higher security. But the execution cost is increased. The next work should find the balance point of security and performance.

REFRENCES:

[1] T. W. Hou, H. Y. Chen, M. H. Tsai, "Three control flow obfuscation methods for Java software", *IEE Proceedings: Software*, 2006, Vol. 153, No. 2, pp.80-86.

[2] Y. Hiroki, M. Akito, N. Masahide, “A goal-oriented approach to software obfuscation", *Computer Science and Network Security*, Vol. 8, No. 9, 2008, pp.59-70.

[3] M. Nikos, K. Nessim, P. Bart, "A taxonomy of self-modifying code for obfuscation", *Computers and Security*, Vol. 30, No. 8, 2011, pp.679-691.

[4] K. A. Bakar, B. S. Doherty, "An enhancement of the random sequence 3-level obfuscated algorithm for protecting agents against malicious hosts", *International Journal of Computers Communcations and Control*, Vol. 2, No. 2, 2007, pp.159-173.

[5] B. Shlomo, K. Tsvi, R. Francesco, "The impact of data obfuscation on the accuracy of collaborative filtering", *Expert Systems with Applications*, Vol. 39, No. 5, 2012, pp.5033-5042.

[6] A. Hessler, T. Kakumaru, H. Perrey, D.Westhoff, "Data obfuscation with network coding", *Computer Communications*, Vol. 35, No. 11, 2010, pp.1-14.

[7] C. R. Subhra, B. Swarup, "RTL hardware IP protection using key-Based control and data flow obfuscation", 23rd International Conference on VLSI Design, January 3-7, 2010, pp.405-410.

[8] C. S. Collberg, C. Thomborson, "Watermarking, tamper-proofing, and obfuscation-tools for software protection", *IEEE Transactions on Software Engineering*, Vol. 28, No. 8, 2002, pp.735-746.

[9] G. V. Anjaiah, S. Ashutosh, "A neural network approach for data masking", *Neurocomputing*, Vol. 74, No. 9, 2011, pp.1497－1501.

[10] N. M. Karnik, A. R. Tripathi , "Security in the ajanta system", *Software-Practice and Experience*, Vol. 31, No. 4, 2000, pp.301-329.