Scientific Journal Impact Factor (SJIF) : 3.518
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

AN ENHANCED CODE OBFUSCATION FOR CORE JAVA

[1]M.Ashok, [2]N.Dhakshayani,[3]K.Ganga devi,[4]G.S.Siva shankari
[1] Sr.AP/Information Technology, Rajalakshmi Institute of Technology
[2,3,4]Information Technology, Rajalakshmi Institute of Technology

Abstract—Today, Reverse engineering becomes common nowadays. The java source code is compiled to a class file that contains the byte code. Java Virtual Machine (JVM) needs only the class file for execution. The problem is that the class file can easily decompiled into the original source code using java decompiler tools. The best solution to prevent reverse engineering is obfuscation technique i.e. obfuscation of the class file, so that it will be very hard to reverse engineer the original source code. Available obfuscation techniques concentrates more on the identifiers used in the java code, on obscuring the identifiers the code will run without any error and provides the same output as before. This paper demonstrates obfuscation technique that concentrates on both keywords and identifiers used in the java code and obscuring the entire java code and makes the code with unexecutable format.

Keywords— Decompiler, Obfuscation, Code Obfuscation, Reverse Engineering, Machine Intelligence.

I. INTRODUCTION

Reverse engineering of our proprietary applications by unfair competition or malicious hackers may result in highly unacceptable exposures of your algorithms and ideas, proprietary data formats, licensing and security mechanisms, and, most importantly is customers' data. Here is why Java is particularly weak in this respect compared to C++.Obfuscation is a technique to protect against this security threats [10]. The remaining of the paper is structured as section II describes the background process. Section III describes the obfuscation technique used in existing system and its disadvantage [1]. Section IV illustrates the proposed system with enhanced obfuscation technique which provides high level of security for achieving effective software protection. Section V illustrates the experimental analysis of existing and proposed system and finally conclusion is demonstrated in section VI.

II. BACKGROUND

A. Decompiler

A decompiler is a computer program that takes as input an executable file, and attempts to create a high level, compliable source file that does the same thing. It is therefore the opposite of a compiler, which takes a source file and makes an executable. Decompilers usually do not perfectly reconstruct the original source code, and it can vary widely in the intelligibility of their outputs. Nonetheless, decompile remains an important tool in software reverse engineering.

B. Obfuscation

In software development, obfuscation is the deliberate act of creating obfuscated code, making the source code difficult for humans to understand. Developers may deliberately obfuscate code to provide security to the code or its logic, in order to prevent tampering, and stop reverse engineering, or creating a puzzle for someone reading the source code [4]. Developers transform readable code into obfuscated code using various obfuscation techniques.

C. Code Obfuscation

In software development process, obfuscation is the intended act of creating obfuscated code, making the source code difficult for humans to understand [9]. Developers may deliberately obfuscate code to provide security to the code or its logic, in order to prevent tampering, and stop reverse engineering, or creating a puzzle for someone reading the source code in Fig 2.1. Developers transform readable code into obfuscated code using various obfuscation techniques.

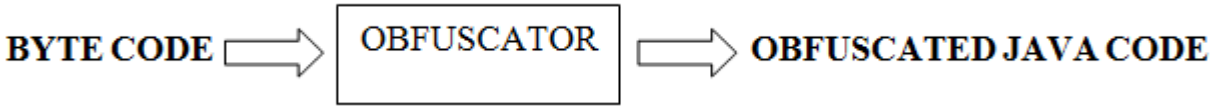

Fig 2.1 Process of code obfuscation

D. Machine intelligence

Intelligence exhibit as artificial (non-natural, man-made) entity , the essential quality of a machine thinking level similar to the same general level of human being is said to be machine intelligence[3,2]. A machine performance is similar to human intelligence [6], such as learning, self-correcting and adapting.

III. EXISTING SYSTEM

In the available obfuscation technique the identifiers are taken more privilege while obscuring with two different levels. Fig 3.1 illustrates the architecture of the existing system for code obfuscation. The aim is to incorporate and enhanced code obfuscating for security features with two levels of security.

i) First, the Java code is given as input to the compiler, the compiler will generate the class file and this class file will undergo first level of obfuscation giving the obfuscated code as output.

ii) Similarly it undergoes second level obfuscation giving more obfuscated code. Then, if the class file is decompiled to get the original Java file in obfuscated version.

The Working function of the proposed architecture as level by level is explained as follows:

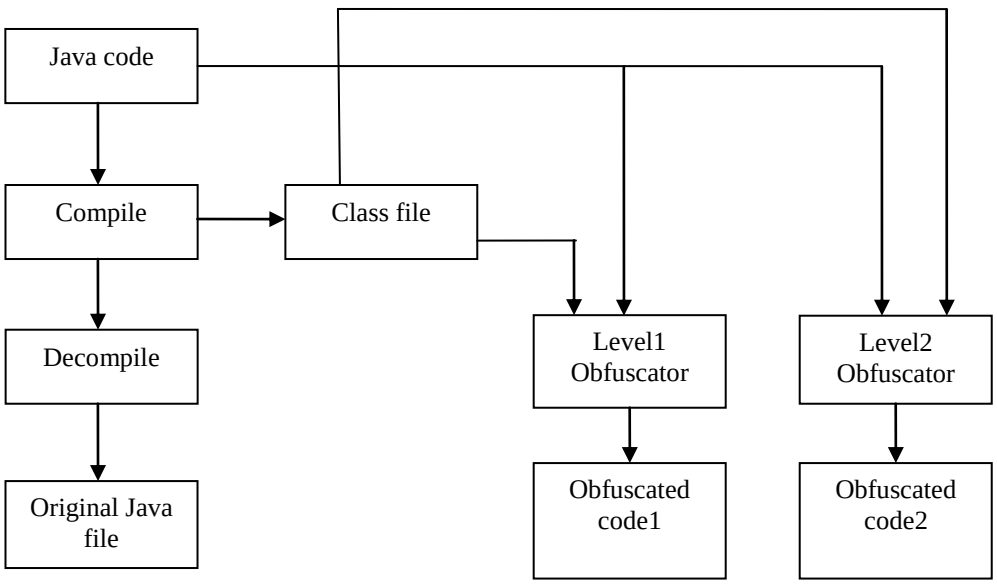

Fig 3.1 Existing system architecture

LEVEL 1:

In this level, the obfuscator finds user defined variable functions and classes then it obscure all the user defined variables with random underscores. Fig 3.2 shows the working function of level1 obfuscator. If the hacker performs reverse engineering process, they will see only the symbols and keywords and not the logic of the code. Though it is difficult to the hackers to find the logic, the security in this level is moderate.

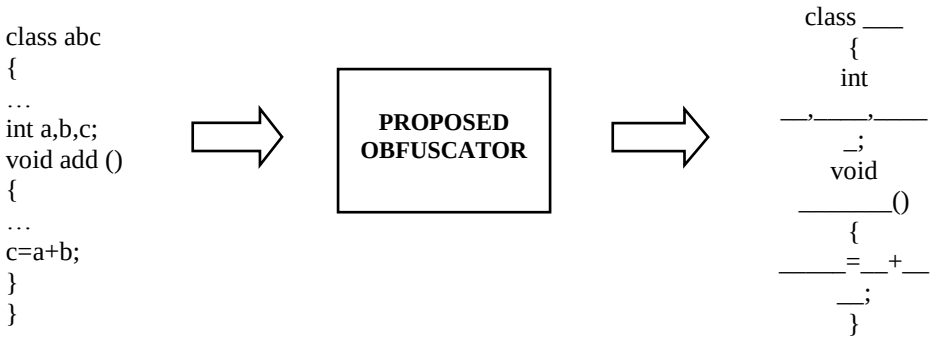

Fig 3.2 Level1 obfuscator function

LEVEL 2:

The level2 is similar to level1 but instead of random underscores, the user defined variables are obscured with the very long randomly operated string with the combination of underscores, numbers and characters. While comparing to level 1 the security level is high. Fig 3.3 shows the working of level 2 obfuscator.

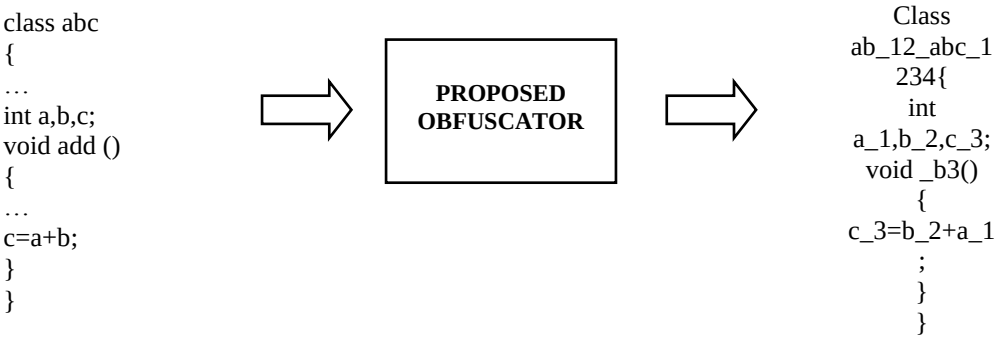

Fig 3.3 Level2 obfuscator function

The drawback of the existing system is code runs without error and displays the same output as before. So, the hacker may get to know some features of the java code. In order to address this problem we came with new solution. This is explained clearly in next section.

IV. PROPOSED SYSTEM

Considering the disadvantage in existing system, we came up with different obfuscation technique to obscure the given java code shown in Fig 4.1. In this technique we take identifiers and also the keywords used in the java code and apply the following steps in the given input file:

Step 1: Split the string token into two half
Step 2: Reverse the two half individually
Step 3: then interchange the reversed splits [5].
Step 4: Repeat step 1, 2, 3 for all the string tokens in the java code.

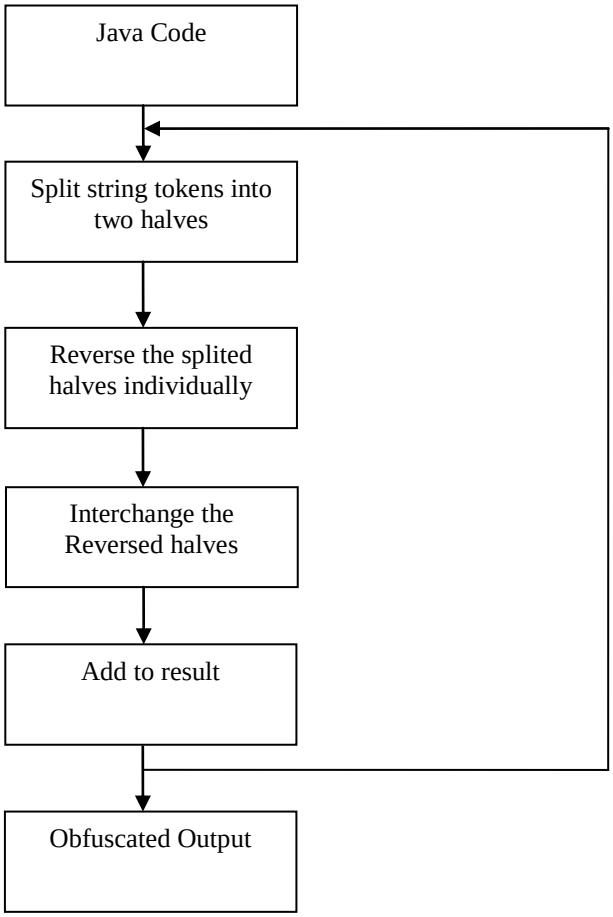

Fig 4.1 Proposed system architecture

LEVEL 3:

In this the security level is very high, by the obscuring the entire program into unreadable form in this the functionality of obscured java file is not same as original. Fig 4.2 shows the working of level 3 obfuscator.

Fig 4.2 Level3 obfuscator function

V. EXPERIMENTAL ANALYSIS

Fig 5 shows the difficulties in finding the source code from the obfuscated version of the code. In the existing system, the probability of finding the logic of the original code is quite easy

because the output of the obfuscated version is similar to the original code so, hacker might get the logic of the code. This is shown using the curve named as existing in fig. In the proposed system, we are going to replace the identifiers and also the keywords used in the source code. So, the probability of difficulty to find the logic of the original source code is high when comparing with the existing system. This is shown using curve named as proposed in Fig 5.

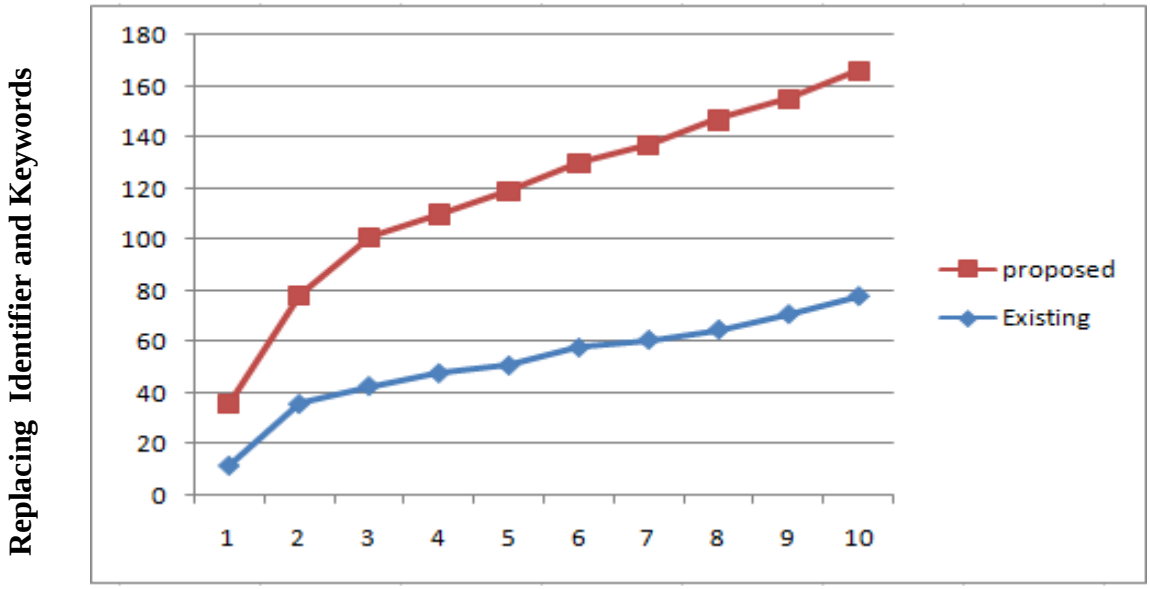

Fig 5: Comparison of Difficulties

VI . CONCLUSION AND FUTUREWORK

Obfuscation is proposed as a PLUGIN for software products. Thus, we protect the class files against the security threats and also provide file level protection. The paper also illustrates the working of a code obfuscator that operates the java files and produces obfuscated versions as output. The existing obfuscation techniques are applied to programs in a static manner. The obfuscation technique is not implemented dynamically. So, as a future work we strive to retrieve to realize our proposed level based approach by implementing dynamic decision making ability to the software. Darwinian evolution shows the behavior of naturally occurring intelligent entities and can be used to guide the creation of entities that are capable of intelligent behavior. Our efforts are directed towards encouraging the development process in software programs.

VII. REFERENCE

[1] N.Dhakshayani, K.Gangadevi, G.S.Sivashankari, M.Ashok, "A Code Level Obfuscator for Core Java," in proc. International Journal of Modern Trends in Engineering and Research (IJMTER), volume:3 Issue: 2 Feb'2016,pg. 349-355.

[2] P. Singh, V. Sidana, K. P. Aggarwal, N. Verma, S. Verma, and A.B.Patki, "Introducing Machine Intelligence quotient as a new COTS Evaluation measure," in *proc. 4th National conference*, INDIAcom 2010.

[3] H.J.Park, B.K.Kim, "Measuring the machine intelligence quotient (MIQ) of human machine cooperative systems".IPSJ Journal, vol .67, no.68, Aug. 2008, pp. 4334-4569.

[4] K. Fukushima, S. Kiyomoto and T.Tanaka, "An Obfuscation Scheme Using Affine Transformation and its Implementation", IPSJ Journal, vol .47, no. 8, Aug. 2006, pp. 2556-2569.

[5] E. Eilam, Revering: Secrets of Reverse Engineering.1st ed. USA .Wiley, 2005

[6] D. B. Fogel and L. J. Fogel, "Evolution and Computational Intelligence," in *Proc. IEEE, Neural Networks*, vol. 4, pp. 1938-1941, Dec. 2005.

[7] G.Naumovich and N.Memom, "Preventing piracy, reverse engineering and tampering."IEEE Computer, vol, 36, no.7.pp.64-71.July 2003.

[8] P. C. Van Oorschot, "Revisiting Software Protection," in *proc. 6th International Conf. Information Security (ISC 03), LNCS 2851,* Springer - Verlag, 2003, pp. 1–13.

[9] C. Collberg and C. Thomborson, "Watermarking, tamper-proofing and obfuscation-tools for software protection," IEEE Trans on Software Engineering, vol.28, n0. 8, pp. 735-746, Aug. 2002.

[10] H. Schildt, *The Complete Reference Java 2*, 5th edition, New Delhi, India: Tata McGraw-Hill, 2002. ch. 17, pp. 546-577.