A Study & Review on Code Obfuscation

Conference Paper · February 2016

CITATION
1

READS
460

5 authors, including:

Saurabh Malgaonkar
University of Otago
28 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Mudit Kapoor
Indian Institute of Technology Roorkee
10 PUBLICATIONS **5** CITATIONS
SEE PROFILE

2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (WCFTR'16)

A Study & Review on Code Obfuscation

Savio Antony Sebastian, Saurabh Malgaonkar, Paulami Shah, Mudit Kapoor, Tanay Parekhji
Computer Engineering Department,
Mukesh Patel School of Technology Management & Engineering,
NMIMS University, Mumbai, India.
savioeven@gmail.com, saurabhmalgaonkar@gmail.com, paulamishah@gmail.com, er.muditkapoor@gmail.com,
tanay_parekhji@live.com

***Abstract*—This paper presents a technical study review of code obfuscation. To address this, we discuss the need and methods of code obfuscation. We analyze the different techniques which are used to thwart reverse engineers and to protect against malicious code injection and attacks. Obfuscation, in software technology, is the deliberate act of creating an obfuscated code, that is difficult for humans to understand. Code obfuscation is a protective mechanism which is used to reduce the attack activities on a software system. It is a behavior preserving program transformation which aims to make a program unintelligible to automated program comprehension tools. Code obfuscation is convenient in situations where depending on cryptographic techniques is not enough; this is normal in remote execution situations where the software is executed on an unforeseen exposed hostile environment, such as the new computing platforms: cloud-computing paradigm and smart phones.**

Keywords—Security, Obfuscation, Code Security, Code obfuscation techniques

I. INTRODUCTION

Information exchange has become an indispensable component in modern society. Vendors provide subject matter to consumers, while consumers interchange information using e-mail, peer-to-peer systems, social networks, or other network applications. The utilization of software applications has become an everyday event of our lives. Obviously, all these applications depend upon the correct functioning of software and hardware system. In the last decade, more and more software is getting widely used over the Internet. Examples constitute from browsers and e-mail clients to games, online shopping e-banking, tax-on-web, and even e-voting. Many wireless devices, including laptops, tablets, and smart phones, are becoming part of our everyday life Naturally, adequate security is essential in this complex heterogeneous environment. There are many threats on the computer system which is worsened by poorly protected software. Therefore, it is important to have an exhaustive threat analysis as well as software protection schemes. The other reason is that the operating systems and the professional applications are very expensive. As an outcome, illegal use of software has emerged. It was found out that protecting software against malicious users is a tough problem. The user is in control of his machine: he has physical access to the hardware, he controls the network connectivity, etc. It became essential that the software owners should manage to arm themselves against these threats. Some examples include popular applications such as Apple's media player iTunes, the IP application Skype, or online games such as World of Warcraft. These applications have been open to attacks since years. Nevertheless, they are still able to withstand the major problems.

It is essential to strength the security system of the computer system. Firstly, software may include secret, confidential or sensitive information, for example medical files or credit card numbers. To guard this data, there is encryption and authentication algorithms, but it is essential to have the secret keys secured somehow. Secondly, we should have the code of applications. It is necessary to have a protective covering on certain software implementation from reverse engineering. And thirdly, there is a need of protection during the program's execution itself, where critical code is executed and some confidential data is accessed. Even when the execution is taking place, one requires to protect the code and data from malicious intents, such as dynamic analysis and tampering.

Thus, the computer industry is focusing on how to protect hosts against malicious programs and developing virus scanners and firewalls. It became essential to do something drastic to stop data from being misused and thus a new concept, code obfuscation was processed. First of all, obfuscation prevents manual inspection of program internals. Secondly, by renaming variables and functions, and breaking down structures, it defends against reverse-engineering. Thirdly, it protects both storage and usage of keys, and it can conceal certain properties such as a software fingerprint or a watermark, or even the location of a flaw in case of an obfuscated patch. However, code obfuscation merely fortifies built-in protection mechanism. In theory, obfuscation could be used to conceal vulnerabilities as well. Thus, security through obscurity is often not thought as a good practice. In the case of obfuscation, the 'key' can specify which transformations were carried out, in what order, and on which section of the code. This key permits the software owner to reconstruct in a deterministic way a user's obfuscated, or even personalized, version of the binary. These keys can be kept in a database until needed for maintenance or analysis of bug reports.

II. NEED FOR CODE OBFUSCATION

A. CIA

Computer scientists define the fundamental security requirements in any system as the following universally known as CIA metaphor:

1. Data Confidentiality: only authorized users can approach and use that data
2. Data Integrity: data cannot be maliciously modified or manipulated with
3. Data Availability: the system must ensure that the data is always accessible when it is needed.

B. Two ways to meet security requirements

1. Open Security (or Security by Design): If the target protected system is open to all, including the attacker , one has to try to make it as strong and fault tolerant as possible. That is referred to as Kerckhoffs' principle [2]; the system security should trust only on its cryptographic key as anything else is public knowledge. Claude Shannon further comprehended this principle, as "The enemy knows the system and assuming he had unlimited computational power, he will crack the system using a brute force attack". In 1976, Diffie and Hellman further polished this idea by taking into account the fact that real attackers are also computationally limited [3].
2. Security through Obscurity: In this approach the designers hide system vulnerabilities in the system implementation (a defense in depth technique). Thus, Code Obfuscation is needed as it is the practice of hiding the purpose, meaning and operation of the code from attackers, either humans or reverse engineering software [4]. This concept is widely common between virus and malware designers to hide their signatures from virus scanners thereby avoiding detection [5].

C. Ways in which it is essential

1. Code obfuscation promotes intensified security by preventing code modifications or 'application hijacking' (the insertion of malicious code).
2. Software developers may also utilize obfuscation techniques to hide flaws and vulnerabilities, or guard intellectual property.
3. Code obfuscation also defend against malicious modifications to a program and software piracy as an attacker must first know and understand the software before they can make specified modifications. The level of security in an application consists of the required resistance of the application against reverse engineering and/or tampering attacks.
4. Open systems are more vulnerable to attacks than closed systems. Desktops, notebooks, or mobile devices are more susceptible to white-box attacks compared to physically shielded servers accessible via a network connection. So it is essential to have code obfuscation.
5. The type of attacks and the amount of resources invested by an attack depend on the value of the content (code or data) and on the nature of the application. Code obfuscation is recommended on content matter that has high value.
6. Content or properties with a longer lifetime require a higher level of trust and security. So it seems logical to have code obfuscation for them.
7. The security of an application can be designed to be periodically renewable. Systems without upgrade possibilities need a higher security level than systems with systematic upgrades. This leads to the importance of code obfuscation for such applications.
8. Global attacks are attacks concerning a whole group of software instances. This is feasible when code contains a "global secret". A single key for all legitimate users or an unguarded security flaw allows an attacker to develop an automated attack and spread it through the Internet. Thus ,once again, the importance of code obfuscation is emphasized.

III. EFFECTIVENESS CRITERIA

A. Based on Complexity Metrics

The measures to evaluate effectiveness of code obfuscation techniques is generally measured from the potency, resilience, cost, stealth [7] in the application.

1. **Potency**: Potency is a measure of how much obscureness is added to the code to make it much more hard to understand to a human attacker. This measures how much complexity the obfuscation techniques increase to the original code in terms of nesting complexity, control-flow complexity, variables complexity and program length.
2. **Resilience**: This metric evaluates the strength of the obfuscation technique with respect to automated de-obfuscators. The following are rules for such assessment:

- This is simpler for local ones and harder for global and inter-procedural predicates.
- Analyzers may compare and correspond obfuscated code to other well-known obfuscation techniques to find common similarities. So obfuscation has to be innovative and not at all generic, that is, more grammatically related to the real program.
- If we have utilized various methods of data and control aggregation and restructuring tools, a program may seem hard to analyze by a human. But for a slicer program, it can carefully work out variable values at every point of the program and which statements lent to that value. In order to obstruct these tools, we can use inherent dependences between variables and add more fake dependences thereby

increasing slice sizes and making it highly improbable for the analyzer to understand them.

- Statistical probability analysis is used to guess the outcome of predicates and evaluate conditional branches. So one has to choose their predicates carefully not to be always true or always false. We have to choose several predicates to be evaluated at the same time, for example, having side effects for opaque predicates.
- Code optimization techniques are used by de-obfuscators that tend to eliminate obfuscations which have been introduced before. So we have to carefully choose what fake code we insert.

3. **Cost**: Cost is a measure of how many additional resources the obfuscated code uses during runtime. The following concepts are included:

- Memory: More memory space will be consumed, especially in the heap, when obfuscation is done
- Storage: There will be problems in storing it or transferring and it may require more bandwidths, as an obfuscated source file size grows larger
- Runtime: Due to code obfuscation, program performance may get degraded. This might be undesirable in certain settings in which applications are sensitively run, for example, real-time systems where the main concern would be meeting execution times constrains

4. **Stealth**: Confusion transform with high toleration is not easy to remove by confusion tools automatically, but it could easily be the attacker's artificial analysis to see through it. In particular, if there are large differences between introduced code by a transformation and the original one, attacker can easily see through it, so every effort should be made to use the syntax similar to the original code structure to strengthen the cover.

B. *Based on Resistance to Attacks*

1. Static Analysis Attack: In this, an attacker builds a CFG (Control Flow Graph, a higher level representation of code) of software that assists to grasp the functionality of software. It is acquired by statically analyzing code [14].
2. Dynamic Analysis Attack: In this, an attacker dispatches the software on various inputs and studies its output by checking the execution traces [14]. Dynamic analysis is more difficult than static analysis as the software needs to be run with different inputs.

3. Code Clone Detection Attack: In this, an attacker finds and removes code clones present in program to reduce code size of program. Attacker can utilize existing code clone detection techniques such as the matching Abstract Syntax Tree (AST) (Baxter et al. [15]) [16].

C. *Based on Program State*

Chow et al. [10], code obfuscation is achieved attained by embedding an instance I of hard combinatorial problem into the program, by applying semantic preserving transformations. This technique expands the state space of the program in such a way that it is difficult to compress the expanded state space.

IV. TECHNIQUES OF CODE OBFUSCATION

Code obfuscation is the process which transforms the source code or intermediate code to make it more difficult to be analyzed. Thus, the reverse engineering of software is more difficult to be implemented. The obfuscation process is implemented in automatic tools called code obfuscation. There are a few important techniques of code obfuscation. There have been great ventures in the field of obfuscation using automated software tools and code transformations. Balakrishnan and Schulze [9] provide a good survey of these techniques. Also, Wroblewski [10] and Collberg etal. [11] provide techniques to transform a code into a functionally equivalent program that makes reverse engineering more complex. In this section we summarize and classify code obfuscation techniques.

Code obfuscation techniques can be categorized according to the transformation subject into various classes:

A. *Layout Transformations*

The first class is layout transformations where the layout of code is changed, removing comments and formatting; sometimes the identifiers are also jumbled.

B. *Control Flow Transformations*

The second class is control-flow transformations where the control-flow of the program is transformed while retaining the same computation functionality. Typical transformation includes aggregation, reordering, including redundant computation.

1. The aggregation transformation changes related computations, and aggregates non related ones. Execution includes adding opaque predicates and variables (with known values to the programmer apriori), loop unrolling, clone methods, and inline/outline techniques.
2. The reordering transformation makes randomization in the order of expressions, statements, and loops, disrupting locality while keeping basic blocks intact. This could be useful when joined with inline and outline techniques above.

C. *Data Abstraction*

The code obfuscation also utilizes obfuscating data abstractions techniques to apply obfuscation. It can be achieved by: name modifications, updating inheritance relations or changing the structure of data arrays.

Modifying inheritance relations: Complexity of program is to be increased by creating partitions among classes or by implementing a dummy class. It is based on the fact that as per the Chidamber metric the quality of the program is increased as the level of inheritance in it increases.

Restructure Arrays: These transformations are of different types such as splitting an array, merging two or more arrays, flattening an array (for example, decrease the dimensions of the array), and folding the array (for example, increase the dimensions of the array)

D. *Obfuscating Procedural Abstractions*

The following section describes the techniques that make the procedural abstractions completely abstruse in a program. Here the original structure of the code gets altered.

1. Table Interpretation: The basic code is changed to a different machine level code which is then interpreted by the virtual interpreter enclosed within the obfuscated code.
2. Inline and Outline Functions: The functions are removed after inlining the code and hence the presence of abstraction is discarded. Outlining is a way in which statements of a function are selected and used to make sub-methods.
3. Clone Methods: The surroundings where the cloning capacities work, assume a vital part in the comprehension of the code.
4. Inbuilt Datatypes: Different customized obfuscating transformations are used to make the data types used in the code unclear and vague.
5. Split variables: There is a division of variables having restricted range into further two or more variables which consists the following:
    - A function to record the values of split variables corresponding to the value of the original variable
    - An another function that records the value of original function into the corresponding values of the split variables.
    - New functions in terms of the necessary tasks on the split variables.
6. Basic Conversions: It involves the conversion of existing possible static data into procedural data.
7. Merging: It involves the process of fusing the available scalar variables into one single variable.

V. ADVANTAGES OF CODE OBFUSCATION

The basic advantage of code obfuscation is the low cost and the flexibility of this technique. Depending on the need for security, an application can be obfuscated accordingly. The extra cost and computation time, presented by the transformations, can be traded off with the performance. The most important advantages of obfuscation are:

A. *Protection*

Code obfuscation protects against static and dynamic analysis attacks as it creates difficulty for the attacker, that is, the attacker requires more time or resources to achieve his goal.

B. *Diversity*

It is possibility to make different instances of the original program, for example, to fight against global attacks.

C. *Low cost*

Obfuscation requires a low maintenance cost due to automation of the transformation process and compatibility with existing systems.

D. *Platform independence*

Code obfuscation transformations can be applied on high-level code so that platform independence is preserved.

VI. DRRAWBACKS OF CODE OBFUSCATION

A. *Performance overhead*

Every obfuscation puts an extra cost in terms of memory usage and execution time needed to execute the obfuscated program.

B. *No long-term security*

Code obfuscation does not render perfect security that makes analysis infeasible. It makes program analysis harder, but does not make it impossible. Barak *et al.* [13] emphasis that a "virtual *black-box* generator", able to hide code of every program, cannot exist. However, this does not mean that protection by obfuscation for some programs is impossible.

C. *Code obfuscation cannot ensure that an algorithm or the source code will never be extracted.*

It rather aims to prevent people or automated programs who can decompile a program within a realistic, specified time.

VII. EXPERIMENTAL ANALYSIS

The following code was tested for its integritity using the Java framework in eclipse tool

Original Code

```
loop = 1;
loopSum = 0;

while (loop <= 100){
        loopSum = loopSum + loop;
        loop = loop + 1;
}
```

A. Layout Obfuscation

```
i=1;sum=0;while(i<=100){sum=sum+i;i=i+1;}
```

B. Control Flow Obfuscation

```
tempFoo = 1;
while(tempFoo != 0){
    switch (tempFoo){
        case 1:
        loop = 1;
        loopSum = 0;
        tempFoo = 2;
        break;
        case 2:
        if (loop <= 100)
            tempFoo = 3;
        else
            tempFoo = 0;
        break;
        case 3:
        loopSum = loopSum + loop;
        loop = loop + 1;
        tempFoo = 2;
        break;
    }
}
```

C. Abstraction

```
loop = 1;
private: tempFoo;
loopSum = 0;

while (loop <= 100){
    tempFoo = loop;
    loopSum = loopSum + tempFoo;
    loop = loop + 1;
}
```

Table 1. shows the integrity of the code against a simple code injection of a script to de-obfuscate the original code. It can be seen that Obfuscation using Abstraction is the best technique in the given scenario.

Code	Security
Original	2 %
Layout Obfuscated	13 %
Control Flow Obfuscated	29 %
Abstracted	76 %

VIII. FUTURE WORK

A. Web Malware

Due to the abundance of web apps, web malwares have considerably increased, thus being the main security threats these days. The developers of web malwares apply the obscurity advancements to make it extremely troublesome for their malware to be dissected. Note that web malwares are for the most part appropriated by misusing web programs' vulnerabilities and malevolent (or bargained) sites. Hence, the present jumbling advancements will be updated for such abuse and adjusted to web environment. Particularly, the jumbling advances for the pernicious JavaScript will be consistently displayed in light of the fact that JavaScript is principally utilized as a vehicle for code injection.

B. Smartphone Malware

Cell phones have made significant achievements in various categories, yet been the most alluring focus for malware developers. Along these lines, it is not astounding that the cell phone malwares are being expanded. When effectively contaminating a gadget, the malware noiselessly sends the gadget proprietor's security data, for example, email, contacts, SMSs, date-books, photographs et cetera to its host machine. It is evident that the confusion innovations will be effectively considered and connected for these malwares. We anticipate that what's more will simply utilizing the present obscurity advancements, malware creators will grow new ones that are vitality and asset proficient, as well as fitting for their objective stage, for example, iPhone or Android.

IX. CONCLUSION

The basic advantage of code obfuscation is the low cost and the flexibility of this technique. Depending on the need for security, an application can It is important to note that code obfuscation is largely an "art" rather than science. The main goal of it is to make attacking complicated enough to repulse attackers, rather than formally proving the strength of algorithms; for example, in games industry, the majority of revenue happens during the first few weeks of releasing new game software. Thus, code obfuscation is used only to resist attacking during such small time window [15]. We are arguing for its utility for the opposite motive of securing cloud-computing environments. In this paper we surveyed the need for code obfuscation to make the resulting code unintelligible for human and hard to reverse engineer or being tampered by automated tools. We also briefly reviewed code obfuscation methods and techniques and quantitatively evaluate their benefits in accordance to a set of reasonable criteria. We hope that continuous study in this field would help to find more defenses that would finally lead to the total endorsement of this new paradigm in our everyday life without any concerns.

X. REFERENCES

[1] T. H. Karas, J. H. Moore, and L. K. Parrott, "Metaphors for Cyber Security," Sandia Report SAND2008-5381. Sandia Labs, NM, 2008.

[2] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding - a survey," Proceedings of the IEEE, vol. 87, no. 7,1999, pp. 1062–1078.

[3] W. Diffie and M. Hellman, "New directions in cryptography," Information Theory, IEEE Transactions on, vol. 22, no. 6, 1976, pp. 644–654.

[4] M. Mataes and N. Montford, "A box, darkly: Obfuscation, weird languages, and code aesthetics," Proceedings of the 6th Digital Arts and Culture Conference, IT University of Copenhagen, 2005, pp. 144–153.

[5] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," Proc. of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010, pp. 297–300.

[6] Borello,J.M..and L.M>E<Code obfuscationtechniques for metamorphic viruses. Journal in Computer Virology, 2008.4(3):p.211-220

[7] C. Collberg, C. Thomborson, and D. Low. "Manufacturing cheap, resilient, and stealthy opaque constructs", In ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL98, San Diego, January 1998.

[8] D. Hyde, ?A Survey on the Security of Virtual Machines.? Dept. of Comp. Science, Washington Univ. in St. Louis, 2009

[9] A. Balakrishnan and C. Schulze, "Code Obfuscation Literature ,"http://pages.cs .wisc.edu/~arinib/writeup.pdf, 2005

[10] G. Wroblewski. General Method of Program Code Obfuscation.PhD thesis, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[11] C. Collberg, C. Thomborson and D. Low, "A Taxonomy of Obfuscating Transformations," Technical Report 148, Dept. of Computer Science, Univ. of Auckland, July 1997,

[12] Karnick, M., MacBride, J., McGinnis, S., Tang, Y., Ramachandran, R. (2006). A Qualitative analysis of Java Obfuscation, Proceedings of 10th IASTED International Conference on Software Engineering and Applications, Dallas TX,USA, November 13-15, 2006.

[13]Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. "On the (im)possibility of obfuscating programs." In J. Kilian, editor, Advances in Cryptology:CRYPTO 2001, 2001. LNCS 2139..

[14] M. Madou, B. Anckaert, B. D. Sutter, and D. B. Koen. "Hybrid static-dynamic attacks against software protection mechanisms", In Proceedings of the 5th ACM Workshop on Digital Rights Management. ACM, 2005.

[15] C. Collberg, and J. Nagra, Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection (1st ed.). Addison-Wesley Professional, 2009.

[16] I. Baxter, A. Yahin, L. Moura, M. S. Anna, and L. Bier. Clone Detection Using Abstract Syntax Trees. In Proceedings of ICSM. IEEE, 1998.

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

[18] 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.

[19] "Java",https://java.com/download, May 2015.

[20] "Eclipse", https://eclipse.org/, May 2015.