International Journal of Engineering & Technology, 7 (4.1) (2018) 22-27

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Code Obfuscation. Where is it Heading?

Asmaa Mahfoud*, Abu Bakar Sultan, Abdul Azim Abd, Norhayati Mohd Ali, Novia Admodisastro

Department of Software Engineering and Information System, Faculty of Computer Science and Information Technology
Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia
* *Corresponding author E-mail:asmaa@idease.net*

Abstract

Reverse Engineering is the process of revealing hidden code from class file. It converts garbage to readable English text. The main purpose of Reverse Engineering is to uncover the hidden code when the documentation is poor, missing source file, and developer is no longer available to provide the original code source file. Hacker uses Reverse Engineering to attack the class file to uncover the code. Then, the code can be reused for other purposes without taking any permission from the original author. The class file contains all the information and business rules that will be revealed once Reverse Engineering process attacks. Anti-Reverse Engineering techniques are developed to stop, delay, and prevent Reverse Engineering; one of the most common techniques is Obfuscation. It has many forms of protection such as, changing the names of classes and variables names, hide classes, and change form of code. In this paper, an appraisal will be conducted to study the current Obfuscation techniques. This research proposes a new hybrid technique that is based on obfuscation; the technique will be using mathematics, Unicode, and unknown language to convert the source file to a garbage running file that does same task which normal source file does for java applications.

Keywords: *Reverse Engineering; Anti Reverse Engineering; Code Obfuscation; Software Security*.

1. Introduction

Reverse Engineering is the process of revealing hidden code and special business rules from the class file of any program developed by java programming language. In the development industry, there is always a fight between the code owners and the hackers, each want to claim the ownership of the code, and only the smart one will win the battel, it doesn't matter if they are the owner of the code or not. for each programming language, there are special reversing tools, for example in java programming language, when a program is developed, it produces **class file** which is compiled to the machine language [1].
The reversing tool that can be used for class file is **JAD** or **Cavaj**, these are the de-compilers that can extract the garbage or byte code from the **class file** to a readable text. It translates java-byte code into readable source code.
Companies nowadays have started to use the advantage of Reverse Engineering to re-do or re-develop old applications and software that were developed in the past, for more than thirty years ago. Reverse Engineering offers a very good advantage where the developer does not have to develop a fresh new software where it is possible to take the old one and just edit on it [2]. This way, the company can save time, money, and even staff. It is obvious that Reverse Engineering salves the problem of obsolete applications, but it has drifted to another path, where the hackers were interested to break the commercial applications and re-use the code for their own benefit, some companies will use the Reverse Engineering to reverse other company's code and edit on it and resell It for its own benefit [3].

2. Current Position of Reverse and Anti-Reverse Engineering

Nowadays developers inherit so many applications and systems, such as banking systems, health systems, vendors systems, switching. These systems are old yet still exist until today because of the Reverse Engineering technology. Software maintenance has become easy and adaptable because of the methods and techniques provided by Reverse Engineering. New type of programmers existed, they are Reverse Engineers, all they do is to break a system or a program by using any of the tools provided by the Reverse Engineering, the purpose of doing this process is to get the code and enhance it or copy it to another similar program, the reason they are doing that is to save time and coast [4].
Reversers have found an easy way to break any system or application and steal the code for their own benefit. Reverse Engineering has opened a lot of opportunities for improvements and for stealing intellectual property, until it went out of control. Due to this illegal process of Reverse Engineering, programmers have come out with Anti-Reverse Engineering tools only when the court is no longer able to protect the intellectual property. Currently in the programming industry there is so many Anti-Reverse Engineering tools and methods available to use wisely [5].

3. Appraisal of the Current Anti Reverse Engineering Tools and Techniques

3.1. Reverse Engineering Tools

The main purpose of Reverse Engineering is to extract information from the class file such as business rules, special code,

classes and methods. There are two main reverse engineering tools, Disassembler and De-compiler.

3.1.1. Disassembler

This tool is used to translate the hexadecimal text into human readable text. IDAPro is used to disassemble hexadecimal text. The main process of this tool is to treat the executable file as a structured object that is created from a database representing the source code [6].
This tool contains FLIRT (Fast Library Identification and Recognition Technology) is an algorithm that contains a file that looks for the program file directory. there will be 2 files which are allowed in the current process, the first one is when the IDA is loaded and the second permit when IDA determines the processor type [7].

3.1.2. De-Compiler

The De-Compiler is known of converting the garbage in the class file to readable text. For instance, if the code is written in java programming language, the reversing tool will be JAD-java or CAVAJ; these tools will generate the actual code from the class file. With Reverse Engineering tools the client can reverse the class file to get the source code and re-use it for their own benefits without permission from the original author. They can get business rules, secret techniques hidden in the code. Then it is difficult from the author to claim ownership to the code [8].

3.2. Reverse Engineering Tools

The purpose of developing or creating Anti-reverse engineering techniques is the obvious danger faced by many companies from illegal Reverse Engineering, security issues and piracy have increased, and the threat of losing the ownership of the intellectual property. Due to the problem occurred because of illegal use of Reverse Engineering, many tools have been developed to stop it or at least delay it for some amount of time. Following is some of the tools to prevent Reverse Engineering [9].

3.2.1. Code Packing

This technique is very common and well known for protection. In code packing, a transformation of the original code to be packed and compressed or encrypted. The packed code will be associated with a restoration routine that is used to trigger the original code and setting. During the process of packing, the code will pass by several. First step, initially, the code will be stored in the memory to get packed, second step is to be transferred into another memory which can be called buffer, and finally get registration for unpacking.
Code packing did not prove itself strong enough to protect intellectual property. Hackers have developed a tool that is called unpacking tool that can unpack the packed code using the registration routine. Unpacking tool can get the hidden binary code. Most known tool for unpacking is PEID, it uses pattern matching approach to check the executable file with signature database to define the packing method used to pack the code before creating executable file [10].

3.2.2. Code Obfuscation

This method transforms the look of the code to make it puzzle. It transforms the code into a complex structure that is difficult to understand. Renaming the class file, methods, and variables is one of the common methods used in obfuscation. Renaming may mislead and confuse the reverse while trying to get some sense of the code. This technique is applied in the source file. Reverse Engineering process does not care about the name of the file, or class or variable, the reverser can always rename the renamed classes and files and variables to get some sense of the code, the java IDE usually provides automatic renaming tool where the reverser can use it to rename all the names needed to be renamed [11].

3.2.3. Binary Level Obfuscation

This technique is based on three steps of code conversion. First step, the instruction of running code is replaced with binary level control. Second step, during the running time, there will be restoration and re-obfuscation to change the original code. Third step is inserting additional binary level of instructions. This binary level will be inserted before the obfuscated instruction. Due to the adding of obfuscation level, when the static analysis runs on the obfuscated code, run time errors will occur. However, the address of the controlled instructions is still available which means the obfuscated code can be revealed, after that the dynamic analysis with step by step execution can reveal the original code [12].

3.2.4. Design Level Obfuscation

The purpose of Design level obfuscation is to prevent dynamic analysis. It depends on three procedures to insure protection. Server, Client-Start-Up, and Client. In this design, the server and client will have to interact with each other via a location of a shared memory, which will be created by the server. The client will execute as a zombie to prevent the debugger from dynamic Reverse Engineering, this process will force the attacker to do static analysis. Since the client is executing as a zombie, it is not possible to present itself as INIT process which is hard for the debugger to analyse. This technique supports parallel processing systems if and only the client code is written in parallel. Otherwise this technique will affect the performance of the system. The performance will be highly reduced. This technique does not suite and not compatible with parallel processing where the code and data are shared between multiple threads [13].

3.2.5. Design Level Obfuscation

This technique works on source file where the developer must change the arrangement of the code to confuse the attacker while reading the code. There are quite many transformation techniques as follows;

3.2.5.1. Stealth

Highly transformed code and high level of confusion is done to the original code, where the attacker finds it difficult understands. However, the advanced reversing tools can easily analyze the code and create readable text [14].

3.2.5.2. Layout Transformation

Remove comments, change overall layout of the code, and jumble the identifiers. Still the reversing debugger can analyze and create better arranged code. [15]

3.2.5.3. Control Flow Transformation

This concept includes aggregation and re-ordering and redundant computation.

a. The aggregation process will change the computations, while execution of the code, an opaque predicates and variables with known values to the original code.
b. The reordering process will randomize the order of expressions, statements, and loops while keeping the basic blocks integrated [16].

3.2.5.4. Data Abstraction

This concept does name modifications, updating inheritance relations and changing the stricter of data arrays. Creating dummy classes and increasing the complexity of the program. It is based on increasing the level of inheritance. This method is going against one of the software quality factors, which is decreasing the

complexity to ease editing and updating. This concept trades readability quality factor for protection [17].
Data abstraction modifies the data structure that is used in the program. It increases the complexity of the code by changing the operations of arrays, stacks, queues and other data structures contained in program [18].

3.2.5.5. Code Shield

This technique targets the class file. It controls the flow of the code, reduces the size of the executable file, and supports all standard and enterprise applications developed by java. This tool is meant to easily obfuscate names of classes and easily control the flow of the program [19].

3.2.5.6. Lexical Obfuscation

According to (Zhang, X.et.al, 2008), there are several techniques and methods to apply obfuscation on the source file such as Lexical obfuscation, Layout obfuscation, Data obfuscation, Control flow obfuscation, Inter-classes obfuscation.
This type of obfuscation is used to change or remove information of the lexical of the compiler and debugging information, comments, and scramble identifiers from the byte code. If this technique is used alone, it won't be strong enough to protect the code; it must be used with some other techniques to guarantee protection [20].

3.2.5.7. Layout Obfuscation

The Layout Obfuscation aims at the inheritance of the classes and methods in the source file. The purpose of this technique is to complicate the business rules and logic. In the layout Obfuscation, there is adding more operators and operations along with the code to make the reading difficult [21].

3.2.5.8. Control-flow Obfuscation

The purpose of Control-flow obfuscation is to change the logical control structure of the statements of the code. The statements that will be changed are loops and conditional structure. This method modifies the actual flow of loops and shows the virtual flow which performs the same operation as the real loop. This change will be in the source file. One must note, reading the code after this change will be difficult [22].

3.2.5.9. Byte Code Obfuscation

When a program is developed in java a byte code will be generated, the byte code can be called the mirror of the actual code. It contains characters and symbols from the original source code. Reverse engineers or crackers have exposed the secrets of the byte code that's why it is easy now to develop software in such a short time, but this will be called illegal development because there is no permission from the author granted [23].
Some companies hire reverse engineers to break up license of their developed software for them to increase the security. The source code and byte code obfuscation are the most effective techniques to be used to protect the original code and prevent the threats of reverse engineering.
SCO is most known technique to replace all the class of software with ambiguity, such as to replace names of classes, methods, and variables with the unclear names, to make it hard to understand, if the reverser tries to read. BCO-Byte code obfuscation is a technique that replaces the original class files another class files that are difficult to decompile and re-compile [24].

3.3. Java Obfuscators Tools

There are many tools that are used to prevent Reverse Engineering. The tools usually protect the executable file. Below are some of the common tools used in the industry;

3.3.1. Zelix Klass Master

This is an obfuscation tool that reduces size of the byte code. It renames the actual names and changes the flow of the code. It does stack trace translation and change the logs [25].

3.3.2 Y-Guard

This tool is used to protect the source file from possible threats that may attack the source code. The obfuscation technique that is used by this tool will replace the package, class, method, class, methods, and field names with some random characters that are difficult to understand. Also reduces the size of the class and jar [26].

3.3.2. Pro-Guard

This tool does obfuscating and optimization. It is free to use for java class file. However, it is like other tools to reduce the size of the file that contains the byte code after obfuscating. This tool looks for the unused classes, fields, methods, and attributes in the source file. It also optimizes the Java byte code and shortens all unused instructions. It can rename the classes, methods, and fields to protect your source code [27].

4. Limitation of the Obfuscation Techniques and Tools

Most of the tools and techniques are aiming to change the flow of the code or rename the classes and methods. Or apply protection on the class file by hiding the byte code. Reverse Engineering tools are very strong to rename again the classes and methods to understandable names, if they don't do this feature; the current IDE for java does the renaming easily, so the cracker can make some sense of the code. The Reversing tools can also get the registration file, reveal hidden business rules and rearrange again the flow of the code [28].
Most of the Anti-reverse engineering tools seem to focus on source file which is good factor, but the protection that they are using is not strong enough to stop the illegal use of Reverse Engineering, whereas, they focus on changing the flow of the code or renaming the classes and methods, or adding some unused comments or compressing the size of the source file or pack the code with some registration key to trace [29].
The reversing tools that can unpack, rename, change the flow and create a code that makes sense for the reader to understand. There are some other ways of protections such as limit the time of using the program or allow few times to use the program. After obfuscating the code in the source file, it is still containing some debugging information in the byte code; this information can be used by the reversing tool to trigger the rest of the code. According to the review, until today there is no obfuscating tool that has been proved to be the best tool of protection [30-31].

Figure 1 presents the limitation of various tools and techniques; the below figure discusses the limitation and weakness of Registration and serial numbers.

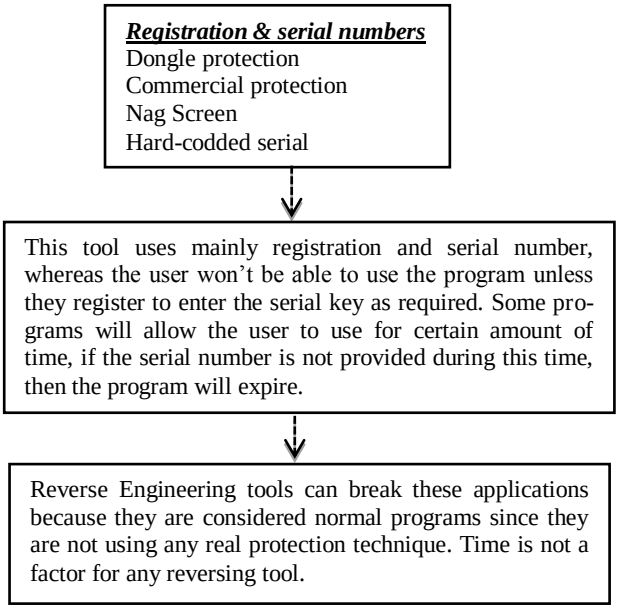


Fig 1. Anti-reverse engineering strategies

Figure 2 presents various tools and techniques of obfuscation; the below figure discusses the limitation and weakness of the obfuscation tools it terms of defense.

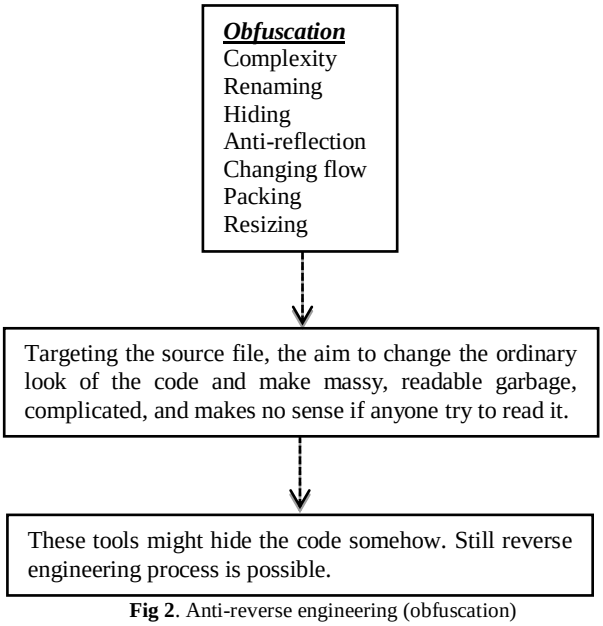
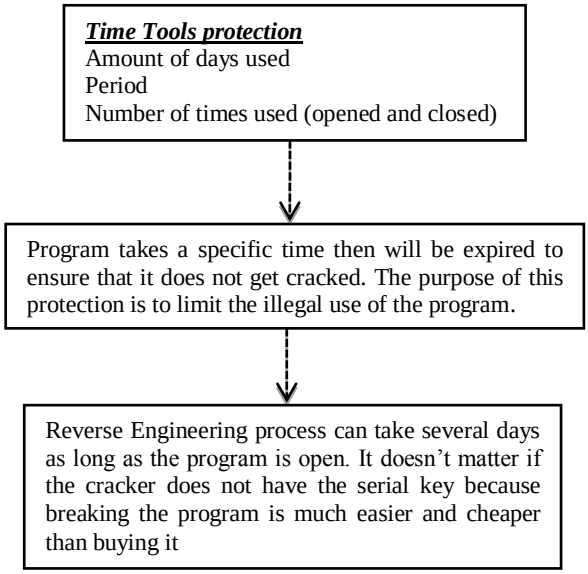


Fig 2. Anti-reverse engineering (obfuscation)

Figure 3 presents various timing tools and techniques; the below figure discusses the limitation and weakness of the timing tools and techniques terms of defense against the prohibited reverse engineering



The previous techniques and tools to prevent Reverse Engineering. They do don't lock the program itself, as the Reverse Engineering tools are able to break and read the code. The Anti-Reverse Engineering tools are quite weak when it comes to protection. They are not trying to block the process of Reverse Engineering. The aim of this research is to implement a technique that is based on an algorithm which can block the illegal Reverse Engineering process, the algorithm will be applied in the source file.

5. Contribution

The aim of the current research is to implement a hybrid technique or algorithm that uses obfuscation of the source file. It not prevents the Reverse Engineering, but it prevents Reading, in fact an error should occur during the reversing process. The technique works on converting the code into three levels of conversion.

First level is to convert the English written code into Unicode to ensure that the beginner reverser does not read the actual words.

Second level is to use mathematics to convert the messages into garbage where the reverser and compiler won't be able to read them from the source file or the class file.

Third level is to convert the names of variables and classes into a readable garbage by the compiler but not the reverser. Finally, when compiling the code, the garbage will be converted to garbage, which means, the program will go through two levels of garbage conversion.

Following code is a sample of code before and after obfuscating.

\u0063\u0068\u0061\u0072\u0029\u0028 e09011c / 2 \u002B 17 - 2 \u0029\u0029; \u007D \u0074\u0072\u0079 \u007B a001001a= input.nextLine\u0028\u0029; \u0069\u0066\u0028 a001001a.equals\u0028""\u0029\u0029 \u007B \u0066\u006F\u0072(\u0069\u006E\u0074 u767d: ",º¬¤É¬ ¾Ë-Æ®´ÆÉË¾Ì¼¸-Æ¸Á¾ËºÂ¤Î¬´Ë ¸º¤¾¸".\u0074\u006F\u0043\u0068\u0061\u0072\u0041\u0072 \u0072\u0061\u0079\u0028\u0029\u0029 \u007B \u0053\u0079\u0073\u0074\u0065\u006D.\u006F\u0075\u0074.\u0070\u0072\u0069\u006E\u0074\u0028\u0028 \u0063\u0068\u0061\u0072\u0029\u0028 u767d /2 \u002B 17 n101010r1=Integer.parseInt\u0028 a001001a\u0029; \u007D \u007D \u0063\u0061\u0074\u0063\u0068(Exception e) \u007B \u0066\u006F\u0072\u0028\u0069\u006E\u0074 i878g:", º¬¤É¬ ¾Ë-Æ ¾Ì¼¸-Æ ®ÁÆ¼¤Ë Á¾ºÕ ¾ÁË ¤º \u0076\u006F\u0069\u0064 \u0069\u006E\u0074 g1111g:"¾Ë-Æ É¬Á¾ë ¾Ì¼¸-Æ ¾Ì¼¸-Æ".\u0074\u006F\u0043\u0068\u0061\u0072\u0041\u0072\u0072\u0061\u0079\u0028\u0029\u0029 input.nextLine\u0028\u0029; d10010d:", º¬¤É¬ ¾Ë-Æ É¬Á¾ë ¾Ì¼¸-Æ ¸Á¾Ë º¬¤Î¬´Ë¸º¤¾¸".\u0074\u006F\u0043\u0068\u0061\u0072\u0041\u0072\u0072\u0061\u0079\u0028\u0029\u0029 \u0069\u006E\u0074 c101c:", º¬¤É¬ ¾Ë-Æ ¾Ì¼¸-Æ ®ÁÆ¼¤ËÁ¾ºÕ¾ÁË¤º, ²¤¸¬Ë´¤º".\u0074\u006F\u0043\u0068\u0061\u0072\u0041\u0072\u0072\u0061\u0079\u0028\u0029\u0029 s110110d:"Ë²- ÉÌ¼ ´É a001001s:"Ë²-¸´Î´«- ´É d110d:"Ë²- ¼Ì°Ë´ ´É \u007B

The above code is only the first trail sample to proof the concept of the research. As seen, it contains a Unicode, garbage code in the source file. Only two levels are used in this sample of code. Figure 4 presents the outcome of running result after reversing the obfuscated code by using **JAD**-Java reversing tool

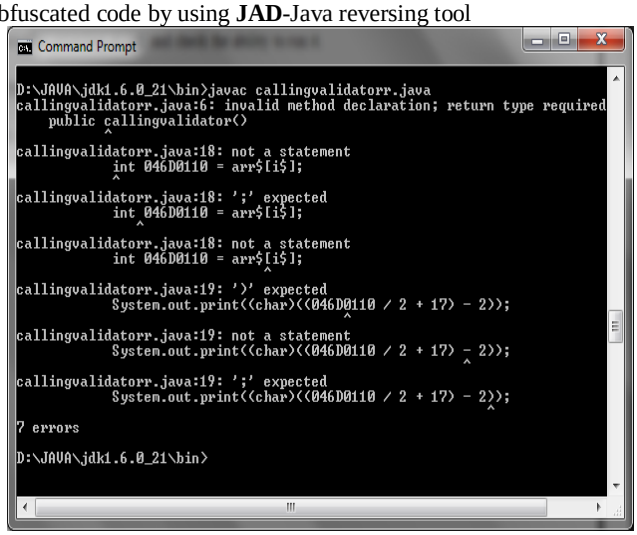


Fig 4. JAD test part 1

Figure 5 presents the outcome of running result after reversing the obfuscated code by using **JAD**-Java the result of part 2 test using JAD against the same class file

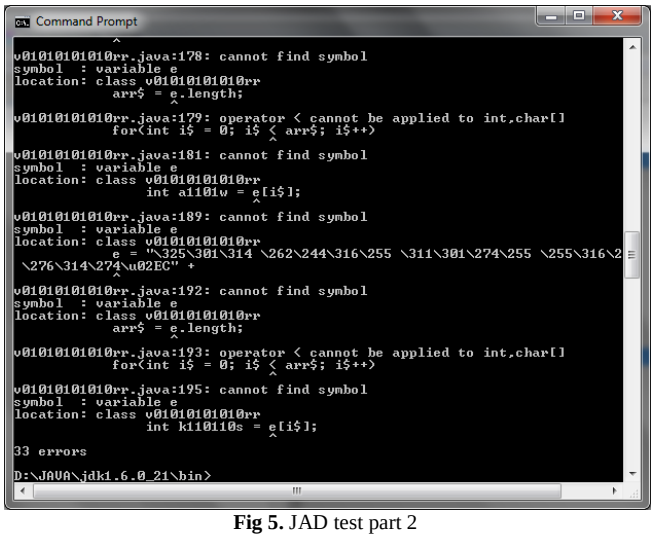


Fig 5. JAD test part 2

6. Conclusion

The prohibited use of reverse engineering has created a lot of cases in the court. It has happened many times that companies stand against each other front of judge in the court and fight the ownership of code, when the fight is taking so long, and none is able to prove for sure the ownership, the judge feels confused and has now ability to decide the ownership of the code claimed. Moreover, only programmers and IT companies understand the programming concepts and principles, which mean a judge who has knowledge only in law and giving the fact only the company with good lawyer, will win the case, it is difficult to fight for ownership. For this reason, developing a good Anti-Reverse Engineering technique is very important to protect intellectual property and keep the ownership.

The developed technique in this research is based on three elements, mathematics, and Unicode and garbage code. Classes, variables and methods names will be changes using the mathematics and the garbage code. the complication of this technique is to ensure that reverse and machine won't be able to the language after reversing.

Acknowledgement

We acknowledge that this research received support from the Fundamental Research Grant Scheme FRGS/1/2015/ICT01/UPM/02/12 that is awarded by Malaysian Ministry of Education to the Faculty of Computer Science and Information Technology at Universiti Putra Malaysia.

References

[1] Acknman, L. (1992). After accolade: time for new laws? washington: Washington Unimsiq.
[2] Briand.L.C, (2006), 13th Working Conference on Reverse Engineering Proceeding, Canada, The Experimental
[3] Paradigm in Reverse Engineering: Role, Challenges, and Limitations, Carleton University,Pp(1,2)
[4] Chiba, S. (2000). Load-time structural reflection in java. Japan, Institute of Information Science, and Electronics, IEEE, Pp. (1, 5).
[5] George, N. et. Al (2000), Reverse engineering anti-cracking techniques. http://www.astalvista.com, the hacking and security community, IEEE. P.(3)
[6] Hardik,S. (N.D), Software Security and Reverse Engineering, IEEE Pp. (1,2, 4, 5)
[7] Hausi, A.et.al. (1994), Understanding Software Systems Using Reverse Engineering Technology, IEEE Pp. (1, 2).
[8] Al-Hakimy, A.M., Sultan, A.B.," Hybrid Algorithm to Protect Java's Code from Reverse Engineering" International conference on software and computer applications (ICSCA), 2015

[9] Himangi,G.(2010), code protection and obfuscation of .Net software using obfuscator, Accessed on January 13, 2018, Available at http://www.dotnetcurry.com/ShowArticle.aspx?ID=542&AspxAutoDetectCookieSupport=1

[10] Iman, M, & Ishaq, A. (2010). Anti-reversing as a tool to protect intellectual property. UAE, IEEE, Vol (5) (1) Pp. (1, 2).

[11] Jared, N. (2008). Rate of software Piracy vs. Malware infection. Debunking bsa's piracy-malware. Accessed on January 13, 2018, Available at http://www.myce.com/news/debunking-bsas-piracy-malware-link-21041/

[12] Linn.C, Debray.S, 2003, Obfuscation of Executable Code to Improve Resistance to Static Disassembly, Washington, University of Arizona Tucson, PP.(2,4,5).

[13] 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.

[14] A. Viticchie et al., "Assessment of Source Code Obfuscation Techniques," 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), Raleigh, NC, 2016, pp. 11-20.

[15] A. K. Dalai, S. S. Das and S. K. Jena, "A code obfuscation technique to prevent reverse engineering," 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, 2017, pp. 828-832.

[16] S. A. Sebastian, S. Malgaonkar, P. Shah, M. Kapoor and T. Parekhji, "A study & review on code obfuscation," 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), Coimbatore, 2016, pp. 1-6.

[17] Min.G.K, et.al, (2007), Renovo: A Hidden Code Extractor for packed Executables, Carnegie Mellon University, IEEE, Pp. (4)

[18] Memon, Jan.et.al, (2006) Emerging Technologies, 2nd international online conference, Preventing Reverse Engineering Threat in Java Using Byte Code Obfuscation Techniques13-14 November 2006, Peshawar, Pakistan, IEEE, P. (690).

[19] Madou, M.et.al, 2006, On the Effectiveness of Source Code Transformations for Binary Obfuscation,P.(4,3)

[20] Zhang, X.et.al, 2008, An Inter-Classes Obfuscation Method For Java Program, International Conference on Information Security and Assurance, Changchun, 130012, P.R.China, IEEE, Pp (2, 4, 6).

[21] Patel, V. (2009). Protect Java code from decompilation using Java Obfuscator. January 13, 2018, Available at http://viralpatel.net/blogs/2009/07/protect-java-code-decompilation-using-java-obfuscator.html

[22] Peikari.C .et.al, (N.D),O'REILLY Media, Know Your Enemy, IEEE, Pp. (10,11,19)

[23] Raymond. J.et.al, (2008), A Survey of Reverse Engineering Tools for the 32Bit Microsoft Windows Environment, Vol.(V,NO.N), Pp.(1,2,3).

[24] NationMaster,(2010). Software piracy rate (most recent) by country. Crime statistics. Accessed on January 13, 2018, Available At http://www.nationmaster.com/graph/cri_sof_pir_rat-crime-software-piracy-rate.

[25] LogicNP, Software. (2010) Crypto Obfuscator For.Net. The power of components. January 13, 2018, Available at http://www.ssware.com/cryptoobfuscator/features.htm

[26] Configure Proguard and DexGuard (Access on 20 September 2016) https://docs.fabric.io/android/crashlytics/dex-and-proguard.html

[27] ProGuard Manual. (Access on 25 September 2016) http://proguard.sourceforge.net/manual/

[28] P. Kanani, K. Srivastava, J. Gandhi, D. Parekh and M. Gala, "Obfuscation: Maze of code," 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA), Mumbai, 2017, pp. 11-16.

[29] Al-Hakimy, A.M., Rajadurai, K.P., Ravi, M.I., "Formulating a Defensive Technique to Prevent the Threat of Prohibited Reverse engineering", International Conference and Workshop on Current Trends in Information Technology (CTIT), 2011, pp. 82-85.

[30] M. Kim et al., "Design and Performance Evaluation of Binary Code Packing for Protecting Embedded Software against Reverse Engineering," 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Carmona, Seville, 2010, pp. 80-86.

[31] M. Kim et al., "Design and Performance Evaluation of Binary Code Packing for Protecting Embedded Software against Reverse Engineering," 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, Carmona, Seville, 2010, pp. 80-86.