Jurnal Teknologi

Full Paper

A SOFTWARE FRAMEWORK FOR CODE SECURITY USING M-COT-METRICS BASED CODE OBFUSCATION TECHNIQUE

R. Senthilkumar*, Arunkumar Thangavelu

School of Computing Science and Engineering, Vellore Institute of Technology, Vellore, India

*Corresponding author
rsenthilkumar@vit.ac.in

Article history
Received
14 July 2015
Received in revised form
28 August 2015
Accepted
15 January 2016

Abstract

Programming security is a paramount concern in IT industry because of its immense monetary misfortunes. Programming is inclined to different security assaults, for example, Software piracy. In this proposal, program security assurance through code Obfuscation, a technique which opposes reverse engineering attacks. In this paper, different sets of criteria are depicted to gauge viability of code obfuscation, for example, intensity: trouble for human to comprehend code, imperviousness to computerized piracy. A large portion of the current obscurity procedures and plans fulfil just a couple of these criteria. In this paper, it shows that the novel code obfuscation plan created for securing exclusive code. A software framework for providing software security using **M**etric based **C**ode **O**bfuscation **T**echniques named as **M-COT** is designed to propose which will maximize the objectives. The essential thought is to change unique code to obfuscated codes which will concede more state space. This is attained by developing obfuscated non inconsequential code clones for intelligent code parts exhibit in unique code. These code clones that are connected utilizing element predicate variables to present legitimate control flows. The performance of the system is observed by experimentation on a couple of projects (for example, scientific calculator code, searching) to assess our plan. The demonstration made that product unpredictability nature of obfuscated code is higher than that of unique code and comparing to single execution Despite of the fact that the proposal builds the improvement of obfuscated code (because of development of non-inconsequential code clones for legitimate code parts).

Keywords: Piracy, Obfuscation, M-COT, Cyclomatic Complexity, Reverse Engineering

1.0 INTRODUCTION

Securing the licensed innovation contained in programming against unapproved access has been a focal issue in System security. The malevolent host issue is an exceptional instance of this general issue. In a malevolent host issue, customer code is sent to and executed in a vindictive host. So such a host can pirate the customer codes each way it can and there is nothing the manager of customer code can do to stop such an assault. Conceivable pirate incorporate programming theft, noxious figuring out piracy and altering piracy [15]. In a programming theft, a pirate makes unlawful duplicates of customer code and after that exchanges them. Pirates can likewise figure out the customer code to acquire some mystery data contained in the code which has a place with the manager of code just [12]. For instance, the customer code may contain a calculation that will provide for its manager edge over his rivals and subsequently is a prized formula. His rivals can uncover this mystery by means of figuring out piracy. Altering piracy alludes to an assault in which an assailant concentrates, adjusts, or something else mess with the mystery data contained in the customer code. In this paper, we will focus on the best way to secure customer code against the second kind of piracy, to be specific the figuring out assault. As specified above, customer code is dispatched to and executed in pernicious hosts. So the holder of customer code does not have any control over what the end host can do with its code. Luckily, in spite of the fact that we are not ready to stop those piracies, there are approaches to make it troublesome enough for those assaults to attain the wanted achievement. Code obfuscation has been the real security instrument against figuring out assaults [1]. To perceive how code obscurity can be valuable, It should be first see how figuring out

functions. A figuring out piracy is fundamentally a process that inverts a bit of programming in its executable double structure go into the source code that it is composed in the code. It inescapably includes examining the control stream and information stream of the project. This data can be procured through static examination of program. "Static investigation alludes to procedures intended to concentrate data from a static picture of a system program." There are different approaches to gain program data, through dynamic examination for instance [12].

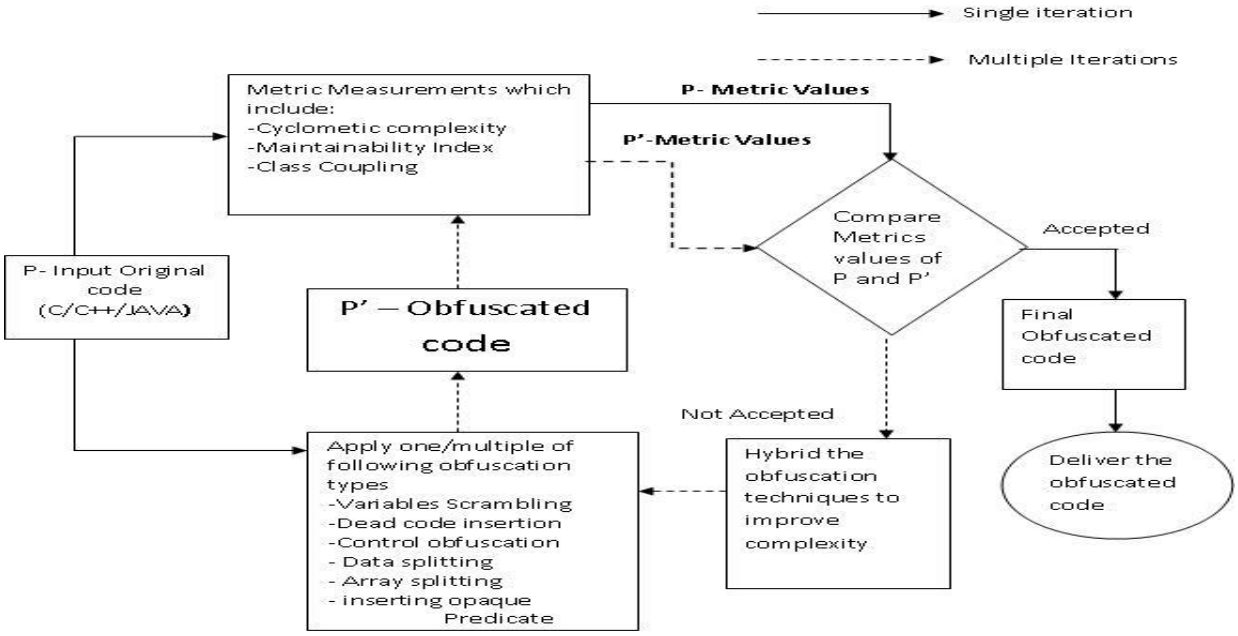


Figure1 Propose Block diagram – **M-COT** (**M**etrics based **C**ode **O**bfuscation **T**echniques)

In this paper, It will limit ourselves to propose **M-COT** (**M**etrics based **O**bfuscation **T**echniques) to manage figuring out the attacks focused around static examination. The flow diagram of M-COT is demonstrated in the Figure 1. M-COT is an iterative system which will apply the Code obfuscation techniques till the complexity level of the obfuscated code is accepted by the developer. Subsequently rendering the changed project confused for a machine mechanized static analyser however practically equal to the first program. Obfuscation changes can be classified as lexical change, control change and information changes.

2.0 LITERATURE REVIEW

The substance of the writing overview into the accompanying subsections. The initially subsection gives writing on key programming security strategies. In the second subsection, it is examine the code muddling strategies and plans. The third subsection presents writing on criteria used to quantify adequacy of code obscurity and writing on de-obfuscation strategies.

2.1 Programming Protection Techniques

Collberg et al. present procedures for securing programming, in particular obfuscation, watermarking and tampering for guarding programming against converse building, programming robbery and altering assaults separately. The paper incorporates subjects, for example, program investigation, static and element code obfuscation, obfuscated hypothesis, software tampering, static and element programming watermarking etc. The system presents scientific classification of code obfuscation systems (for instance, design obfuscated) alongside samples of each one sort of obscurity systems. This paper gives point by point plan for control obfuscation by utilizing opaque predicates. This paper says that in code obfuscation, security can be attained by changing unique code to obfuscated code, such that space of conceivable changes is substantial. This is like security gave by cryptography, that is, security quality relies on upon key-estimate (that is, conceivable encryption decisions). This paper additionally portrays other programming insurance strategies, for example, programming assorted qualities. In software differences, assurance is attained by making different renditions of programming which are practically equal.

2.2 Code Obfuscation Techniques

This paper is discussing on code obfuscation strategies and plans. The main subsection presents methods particular to identifier renaming (a kind of format muddling). The second subsection concentrates on systems particular to control obfuscation.

a) Identifier Renaming Obfuscation

In identifier renaming (or scrambling identifiers) method unique serious names are supplanted with arbitrary negligible names. It is restricted change (as unique significant names can't be recuperated by assailant) without any expense overhead. It is generally backed by business. Experimentation was performed on some sample code. The identifier renaming, assailant's endeavors are expanded to perform effective assault. This paper likewise demonstrates that actually for experienced assailant, it requires at any rate twofold time to complete assault. , if the obfuscated byte code is decompiled, the created code can't be recompiled because of the slips presented.

b) Control Flow Obfuscation

This subsection presents a study on code obfuscation methods focused around change of control obfuscation of a system. Less concentrates on control obfuscation procedure for programming assurance. Control obfuscation changes shroud calculations utilized as a part of projects by presenting new fake control obfuscation and by making gimmicks (for instance, unstructured control obfuscation diagram) at item level

c) Obfuscation Schemes Using Inserting Dead Code

This subsection presents obfuscation which created utilizing blend of existing obfuscation systems and dead code components, for example, utilization of progress toward oneself, time delicate codes and utilization of disseminated framework. A system is ensured by a technique containing numerous time touchy codes which are overwritten by dead codes utilizing changing toward oneself component. In the event that execution time of a watchman code square is inside foreordained extent, time delicate code gets to be unique one. On the off chance that execution time is out of extent, dead code. This methodology opposes element reverse engineering assaults however it obliges exact profiling data, (for example, estimation of time taken by dead code) of unique code before obfuscation it.

2.3 Measuring Effectiveness of Code Obfuscation and De-obscurity Procedures

Collberg et al. depict measures, for example, intensity, strength and expense to assess nature of code obfuscated changes. These measures are focused around programming intricacy measurements, (for example, cyclomatic unpredictability, Mccabe). The proposed a sensible assault model where assailant can investigate each guideline and information esteem at each project point. In this battering approach, dynamic system is utilized to follow execution of unique system and instrumentation data is assembled for assaulting programming. They accept this methodology through a case ponder on programming watermarking calculation. To assess the control obfuscation changes on four programming intricacy measurements: control, control stream, information and information stream. They assess three changes: control stream straightening, static dismantling upsetting and changes focused around opaque predicates. For instance, they demonstrate that control stream straightening expands cyclomatic number (a product many-sided quality metric) as every essential piece is considered for straightening. Investigative measurements are produced to evaluate execution of code obfuscation. For sample, obfuscated measure is capacity of metric parameters: intensity, versatility and taken a toll. This paper additionally observationally assesses different business obfuscator's research utilization of programming obfuscated in building guaranteeing toward one versatile specialist. This paper infers that there is no common obfuscated problem and creators accept that all robotized muddling is simply imitating. They infer that product obfuscated is valuable just in the event that obfuscating official applies an arrangement of change to obfuscated code in which change reliance diagram (demonstrated as limited state automata) is developed.

3.0 OBFUSCATION SCHEMES

In this paper, outline of novel code confusion plan is introduced. Obfuscated code developed by applying our plan fulfils the criteria and furthermore opposes figuring out assaults utilized for measuring viability of code confusion. The fundamental thought is to change unique code to obfuscated code having blasted state space. State space of obfuscated code is stretched by embedding's non trifling code clones developed for legitimate code fragments display in unique code. These non-insignificant code fragments are connected utilizing element predicate variables. An identifier renaming strategy is connected to expand assailant's push to comprehend obfuscated code. Comparing to single way present in unique code, exponential number of legitimate ways is presented in obfuscated code. Any execution way is haphazardly chosen to perform processing, for example, permit checking system. Aggressor needs to execute programming with obfuscated code numerous times to produce execution follows to comprehend usefulness of programming. In this segment, a novel code obfuscation plan is exhibited with a sample of information preparing code. The

obfuscation plan comprises of the accompanying four steps.

3.1 Step-1: Construction of Logical Code Fragments

In the first step, code executing mystery calculation is isolated into coherent code pieces. Every legitimate section executes particular usefulness which known to software developer.. Yet aggressor does not know particular usefulness of these legitimate code parts (as calculation or code is not freely accessible). There are two conceivable decisions for making legitimate code parts. They are as per the following:

a) Utilization of Basic Block

Basic segments are a grouping of code guidelines having single section and passageway focuses. It can't contain control flow directions, for example, restrictive proclamations.

b) Utilization of Methods (or Subroutines)

Methods / subroutine are larger amount representation of code. Techniques have rich processing structure which is spoken to utilizing Control Flow Graph (CFG). In the code obfuscation of essential fragments as a code section is portrayed. Essential pieces can't contain dialect builds, for example, circles or restrictive articulations. Consequently, if consistent code discontinuity is completed at fundamental piece level, restricted system usefulness can be actualized. The obfuscation plan proposes production of coherent code sections at larger amount program representation, for example, techniques (or subroutines). This rich reckoning structure permits designer to develop code parts actualizing particular usefulness. Sensible code sections are developed in such a route, to the point that they are executed in a succession to perform processing.

3.2 Step-2: Construction of Non trifling Code Clones for Fragments

For each one code part, non-insignificant code clones are developed. Non trifling code clones are characterized as a situated of code parts which perform same calculation however they have diverse code structure, for example, Abstract Syntax Tree (AST). This is demonstrated in the Figure 2 with a sample of variable swap usefulness which is actualized utilizing memory and XOR operation.

Swapping using memory	Swapping using XOR
swap(in a, in y) in temp; temp = a a= y y=temp	swap(in a, in y) in temp temp = a ^ y a= temp ^ a y = temp ^ y

Figure 2 Non trifling code clones for variable Swap operation

Because of presentation of code clones, programming intricacy of obfuscated code is expanded. The cloning technique is portrayed as a system to build reverse engineering endeavours. The paper likewise depicts code clone identification methods, for example, matching Abstract Syntax Tree. These can be utilized to catch and uproot code clones, with the goal that figuring out endeavours by aggressor are decreased. There are two conceivable decisions for making code clones to muddle code. They are as takes after:

a) Insignificant Code Clones

Insignificant code clones can be made utilizing identifier renaming obscurity strategy. For instance, set of pseudo code guidelines have same code structure, for example, Abstract Syntax Tree (AST), however they have distinctive variable names.

Swap1: int x, y, t; t = x; x = y; y = t;
Swap2: int a, b, u; u = an; a = b; b = u;

Dead code clones perform same processing however they have diverse code structure, (for example, AST) as indicated in the dead code clones may be distinguished utilizing procedures, for example, matching Abstract Syntax Tree (AST) . However dead code clones can't be discovered utilizing existing code clone recognition methods as code structure, (for example, AST) is diverse for distinctive clones. The accompanying Figure 3 demonstrates the framework in the wake of building dead code clones for sort and inquiry coherent code pieces. For purpose of understanding, diverse calculations are indicated for diverse code clones, in spite of the fact that engineer can build distinctive non inconsequential code clones utilizing same calculation.

Note that rectangular pieces are utilized to speak to a wrapper structure for legitimate code sections (for sort and pursuit usefulness) in the middle venture amid application of the muddling plan. Engineer needs to connection these non-unimportant code clone fragments

3.3 Step-3: Linking Code Clone Fragments utilizing Dynamic Predicate Variables

In this step, non-insignificant code clone parts are connected utilizing predicate variables. Predicate variables are situated of outflows that are assessed to Boolean esteem (either genuine alternately false esteem). There are two conceivable decisions for connecting code clone sections.

a) Utilization of Static Hazy Predicate Variables

Static hazy predicate variables are portrayed by *Collberg et al.* and others. A sample of static misty predicate variable is contingent articulation "b:= (x == x+1)" which dependably assesses to false esteem. Such predicate variables present fake control way in code. Static predicates can't avoid dynamic investigation assaults as fake control ways are never chosen for execution.

b) Utilization of Element Misty Predicate Variables

The predicate variables are a set of related Boolean variable. These variables assess to same esteem in given run of programming yet they assess to diverse qualities at distinctive run of programming. The obfuscation plan utilizes a variation of element predicate variables, a set of conditions which are assessed at run time such that exponential number of substantial blends of estimations of predicate variables is conceivable. These variables empower determination of a specific blend of code clone pieces for a given run of obfuscated programming. These element predicate variables present substantial control stream ways by keeping element structures, (for example, connected rundown) demonstrated in the Figure 4. These element predicates build static and element investigation endeavours as exponential number of code clone blends are workable for execution.

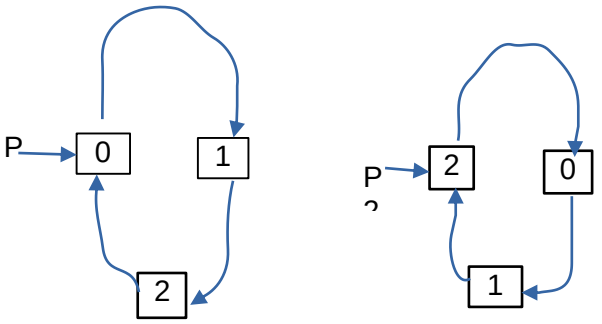


Figure 4 Dynamic Predicate Variables for Sort and Search Fragments

Assume that two connected records are kept up - one for "sort" part and other for "inquiry" section. Variables P1 and P2 move haphazardly through the rundowns such that they point to distinctive hubs (containing whole number information values) at diverse times. For a given run of programming, clones for separate sections are chosen by matching clone identifier (a one of a kind whole number worth allocated to each one code clone of section) with information quality pointed by predicate variables. The Figure 5 demonstrates the information handling framework in the wake of connecting the code clone sections utilizing element predicate variables. Any grouping of intelligent code clone parts is feasible for given run of programming. Aggressor needs to execute programming different times to create follows for comprehension usefulness of programming.

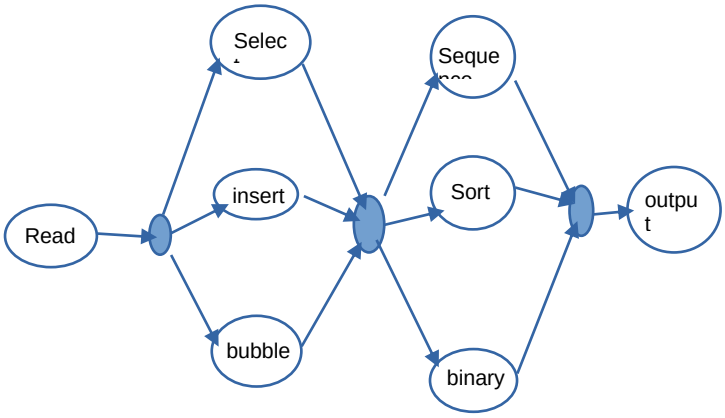


Figure 5 Linked Code Clone Fragments for Data Processing Application

3.4 Step-3: Applying Identifier Renaming Technique

In this step, identifier renaming system is connected to intelligent code clone fragments. In this paper we demonstrate that assailant's endeavours are expanded if code is obfuscated utilizing identifier renaming strategy. In identifier renaming system, serious names, (for example, "sort") are uprooted. It is restricted change (as unique names can't be recouped by aggressor) and it has almost no taken a toll overhead. In spite of the fact that the obfuscation plan proposes utilization of identifier renaming strategy, the plan does not confine utilization of other condition of the craftsmanship code obscurity strategies, for example, control obfuscation, control stream levelling for obfuscating of legitimate code clone sections. The Figure 6 shows obfuscated code in the wake of applying identifier renaming code obscurity strategy. Note that in the figure, just compelling capacity names, (for example, "supplement") are supplanted with arbitrary inane names ("h"), yet in genuine code, all compelling names, for example, neighborhood variables, parameter can be supplanted as well.

4.0 RESULTS AND DISCUSSION

Practicality Index (mi) – Analyses a list worth extent from 0 to 100 that connotes keeping up the code.

Great practicality of the code is distinguished if the quality reach is high appraisals are given to recognize and troubleshoot in your code.

Green imprint - 20 and 100 shows code has great upkeep list (mi).
Yellow imprint - 10 and 19 shows code has modestly upkeep record (mi).
Red imprint - 0 and 9 shows code has low upkeep list (mi).

Cyclamate Complexity – Procedures the essential unpredictability in the code. Cyclamate multifaceted nature is created by figuring the quantity of diverse code ways in the project. A program that has complex control stream will oblige more tests to attain great code scope and will be less viable. Multifaceted nature in the system is high if the stream of control needs more number of trials to meet the great maintenance\e or else it will have low support

Cyclamate complexity(cc)
CC = nl - nn + 2(np)

nl = aggregate of connections in the stream of diagram

nn= aggregate of hubs in the stream of diagram

np = aggregate of withdrew parts of the stream of diagram.

Profundity of Inheritance –it determines the aggregate of class definitions that is reaching out up to the inception of the class progression. On the off chance that the pecking order profundity is high then it is hazardous. So it is important to comprehend where the exact routines and fields to be situated or need a substitution Class Coupling –it figures the coupling to incomparable classes by utilizing the accessible parameters, neighbourhood variables, return sorts, capacity calls, regular instantiations, base classes, interface executions, fields characterized on outer sorts, and characteristic design. Programming is thought to be great when it has tight union and low coupling. High coupling demonstrates a methodology that is hard to reprocess and keep up. Lines of Code – It demonstrates the unpleasant number of lines in the given code. The quantity of line in the code is focused around the IL code, so it is not the accurate include of line the source document. On the off chance that the number is high it may show that a capacity or system is requesting to do part of work and it have to be apportioned. It may likewise demonstrate that the capacity or system is hard to keep up.

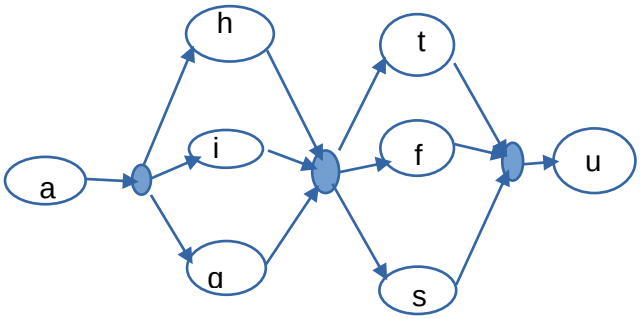


Figure 6 Identifier Renaming for Data Processing Application

4.1 Metric Tool: Source Monitor Measurements

1. Kiviat Chart

This diagram infers that if the greatest furthest reaches of the measurements recognized present inside the red round demonstrates that it has well

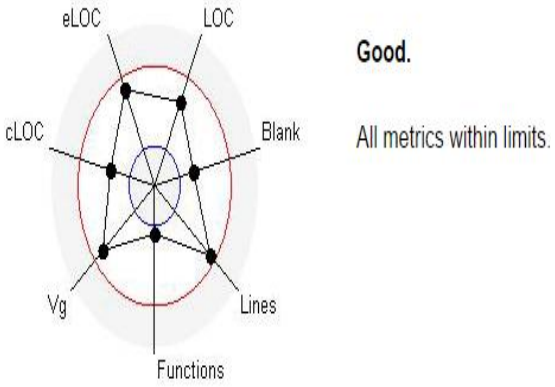


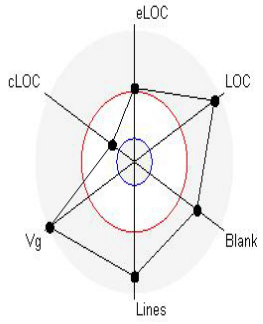
Figure 7 result before obfuscation source monitor

a) Cyclamate Complexity (CC)

Sort/system many-sided quality can be measured by applying cyclamate complexity. It gives better sign of strategy comprehensibility. Every if proclamation, while and for circles switch case || or && inside a contingent administrator.

b) Lines of Code (LOC)

Lines of code (LOC) are a total of each one line that it has any codes. It is cool to quantify however very nearly hard to conclude as is profoundly subject to coding technique, dialect, environment, and coding expertise.



Bad.

All but one (cLOC) of the metrics above high limit.

Figure 8 Result – Source Monitor

5.0 EXPREMENTAL ANALYSIS

Correlation have been made with specimen code prior and then afterward obscurity

a) Before Obfuscation(SOURCE MONITOR)

Test C source code has been composed and its result was broke down in source monitor to check the viability of the code. The acquired result is given

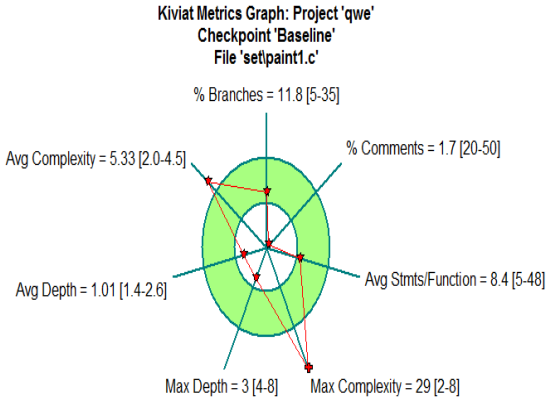


Figure 9 Maximum viability of code

Metrics Details For File 'set\paint1.c' (Printed on 04 Nov 201

Parameter	Value
Project Directory	C:\Users\bv-ar\De
Project Name	qwe
Checkpoint Name	Baseline
File Name	set\paint1.c
Lines	239
Statements	102
Percent Branch Statements	11.8
Percent Lines with Comments	1.7
Functions	10
Average Statements per Function	8.4
Line Number of Most Complex Function	86
Name of Most Complex Function	change_shape()
Complexity of Most Complex Function	29
Line Number of Deepest Block	80
Maximum Block Depth	3
Average Block Depth	1.01
Average Complexity	5.33

Figure 10 Chart shows greatest viability of the given code

Metrics Details For File 'set\paint420.c' (Printed on 04 Nov 2014)

Parameter	Value
Project Directory	C:\Users\bv-ar\Desktop\
Project Name	paint_ofs
Checkpoint Name	Baseline
File Name	set\paint420.c
Lines	247
Statements	108
Percent Branch Statements	13.9
Percent Lines with Comments	3.6
Functions	10
Average Statements per Function	8.8
Line Number of Most Complex Function	94
Name of Most Complex Function	unshape()*
Complexity of Most Complex Function	29*
Line Number of Deepest Block	88
Maximum Block Depth	3
Average Block Depth	1.02
Average Complexity	5.56*

Figure 11 Metrics Result – After Obfuscation

Considering the above result acquired, without influencing the viability of the product, code obfuscation to be carried out so that robbery of the product can be decreased

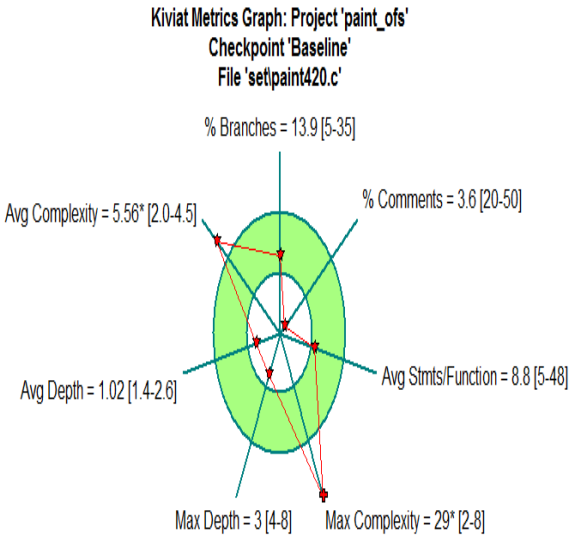


Figure 12 Chart shows still viability of the product is protected

Here obscurity has been carried out by utilizing after systems, for example, immaterial name to the strategies, false remarks, variable disfiguring and so on so perusing and adjusting the jumbled code is truly an extreme part for any coder.

Table 1 Comparison of metrics before and after obfuscation (Source Monitor)

Parameter	Before Obfuscation	After Obfuscation
Lines	239	247
Statements	102	108
% of Branch statement	11.8	13.8
% of lines of comment	1.7	3.6
Avg. statements per function	8.4	8.8
Line number of most complex function	86	94
Line number of deepest statement	80	80
Max. Block depth	3	3
Avg. block depth	1.01	1.02
Avg, Complex	5.33	5.56

b) Before Obfuscation (VISUAL STUDIO-CODE METRICS VIEWER)

Test adding machine code has written in C dialect to ascertain the measurements esteem. Table 1 shows the metrics value that was observed before and after obfuscation. The Maintainability Index of the product is in great condition. Result got by assessing the code is given underneath.

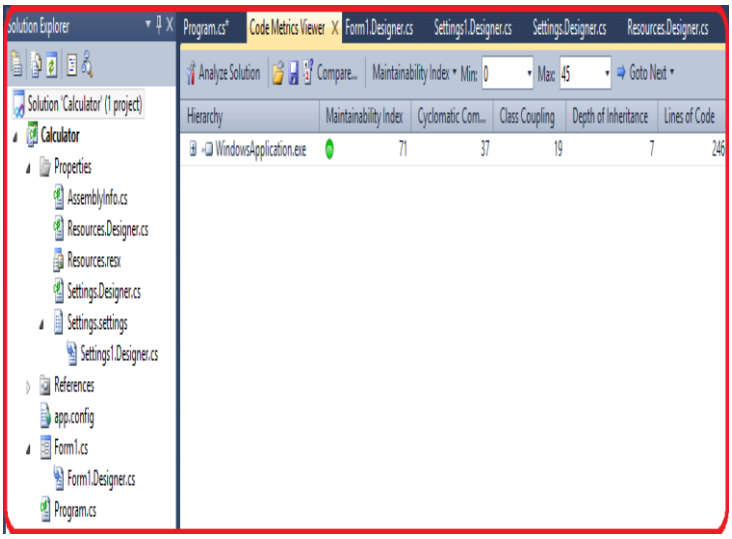


Figure 13 Measurements come about before obfuscation

a) After Obfuscation (VISUAL STUDIO- CODE METRICS VIEWER)

Lines of code for the same system is expanded still the Maintainability Index of the product is stay great. Code is obfuscated and the metrics are still maintained to confuse the pirate in deciding the piracy of the product.

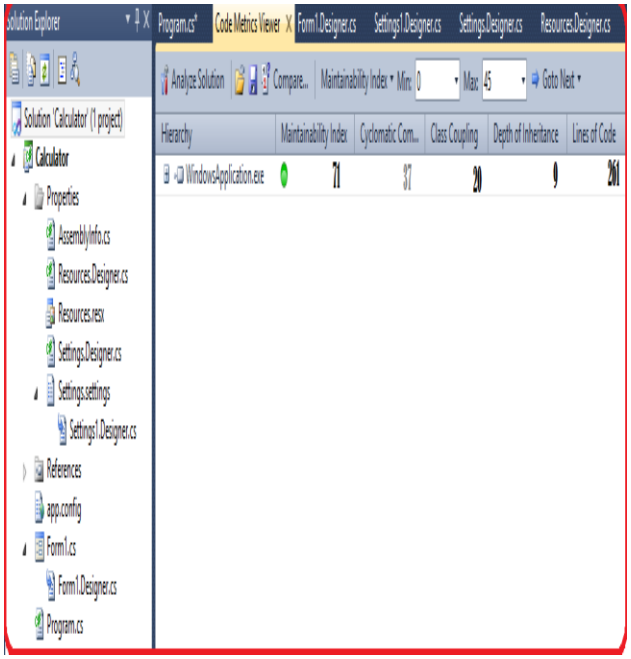


Figure 14 Measurements come about after obfuscation

Table 2 is clearly making us to understand that obfuscation wouldn't increase the dirtiness to code to the maximum level and also it states that obfuscate is

prominent technique for providing the security against the reverse engineering attacks.

Table 2 Comparison of metrics before and after obfuscation

Metric parameter	Before Obfuscation	After obfuscation
Maintainability Index	71	71
Cyclometer Complexity	37	37
Depth of inheritance	7	9
Class Coupling	19	20
Lines of code	246	261

6.0 CONCLUSION

In this paper examined about the results from the source monitor and visual studio measurements viewer tool which will quantify the quality of the code security against robbery the measurements that has used to assess the quality of the security is a quality based measurements the level of muddling may be altered focused around measurements .this may be reached out by including some developed programming measurements with help of the current and the same can be stretched out as dialect free fragments

References

[1] C. Collberg, C. Thomborson, and D. Low. 1997. A Taxonomy Of Obfuscating Transformation. Technical Report 148, Department of Computer Science, The University of Auckland, Auckland, New Zealand.

[2] Christian S. Collberg, Clark Thomborson. 2002. Watermarking, Tamper-Proofing, And Obfuscation: Tools For Software Protection. *IEEE Transactions on Software Engineering.* 28(8): 735-746.

[3] C. Collberg and J. Nagra. 2009. *Surreptitious Software: Obfuscation, Watermarking, And Tamper Proofing For Software Protection*. Addison Wesley Professional.

[4] P. C. van Oorschot. 2003. Revisiting Software Protection. Proc. 6th Int'l Conf. Information Security (ISC 03), LNCS 2851. Springer-Verlag. 1-13.

[5] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. 2001. On the (im) Possibility Of Obfuscating Programs. In J. Kilian, editor, Advances in Cryptology: CRYPTO 2001, 2001. LNCS 2139.

[6] B. Lynn, M. Prabhakaran, and A. Sahai. 2004. Positive Results and Techniques for Obfuscation. In Eurocrypt, Springer Verlag.

[7] M. Ceccato, M. DiPenta, J. Nagra, P. Falcarin, F.Ricca, M. Torchiano, and P. Tonella. 2009. The Effectiveness Of Source Code Obfuscation: An Experimental Assessment. In IEEE International Conference on Program Comprehension (ICPC 2009). IEEE CS Press.

[8] J. Chan and W. Yang. 2004. Advanced Obfuscation Techniques For Java Byte Code. *JOURNAL OF SYSTEMS AND SOFTWARE*. 71(1): 1-10.

[9] Christian Collberg, Clark Thomborson, and Douglas Low. 1998. Manufacturing Cheap, Resilient, And Stealthy Opaque Constructs. In ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL98, San Diego.

[10] Wang, C., Hill, J., Knight, J. C., and Davidson, J. W. 2001. Protection of Software-Based Survivability Mechanisms. In Proceedings of the 2001 conference on Dependable Systems and Networks. *IEEE Computer Society*. 193-202.

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

[12] Falcarin, P., Di Carlo, S., Cabutto, A., Garazzino, N., Barberis, D. 2011. Exploiting Code Mobility For Dynamic Binary Obfuscation. Internet Security (WorldCIS), 2011 World Congress on. 114-120.

[13] Yuichiro Kanzaki, Akito Monden. A SOFTWARE PROTECTION METHOD ASED ON.

[14] TIM. 2010. E-SENSITIVE CODE AND SELF-MODIFICATION MECHANISM. Proceedings of the IASTED International Conference, November 8 - 10, 2010 Marina Del Rey, USA Software Engineering and Applications (SEA 2010).

[15] Christian Collberg. 2011. The Case for Dynamic Digital Asset Protection Techniques. Department of Computer Science, University of Arizona, June 1.

[16] Business Software Alliance. 2013. Eighth Annual BSA and IDC Global Software Piracy Study.

[17] R. Senthilkumar and Dr. Arunkumar Thangavel. 2015. Code Security Using Control Flow Obfuscation with Opaque Predicate. *International Journal of Applied Engineering Research (IJAER)*.

[18] Harsha Varadhan Rajendran, Ch.Kalyan Chandra and R. Senthilkumar. 2010. Design of Java Obfuscator "MANGINS++"- A Novel Tool To Secure Codes. *Journal of Computer and Mathematical Sciences*. 1(6): 636-768.

[19] Vivek Balachandran, Sabu Emmanuel, Ng Wee Keong. 2014. Obfuscation by Code Fragmentation to Evade Reverse Engineering, 2014 IEEE International Conference on Systems, Man, and Cybernetics October 5-8, 2014, San Diego, CA, USA.