Turing Machine Obfuscation Individual Study Report*

Yan Wang†
College of Information Sciences and Technology
The Pennsylvania State University
ybw5084@ist.psu.edu

ABSTRACT

Software obfuscation is a crucial research filed due to the severe computer security situation of Internet era. It could complicate source programs to make it difficult for attackers to do reverse engineering so that malicious codes and software vulnerabilities are constrained from spreading. In this paper we proposed a novel method to do software obfuscation. Predicates and arithmetic operations in source codes are transformed to different Turing Machine programs. Control flow graph could be complicated to a great extent by leveraging this transformation. Software is obfuscated through encoding and replacing partial source program with Turing Machine codes. This paper records the research progress of my research project Turing Machine Obfuscation. In addition, it states our whole research plan and future research steps in detail.

Keywords

Turing Machine; Software Obfuscation; Encoding; Control Flow

1. INTRODUCTION

Software Obfuscation is an important cryptographic concept with wide applications[8].However until recently there was little theoretical investigation of obfuscation, despite the great success theoretical cryptography has had in tackling other challenging notions of security. The goal of conducting software obfuscation is to hide those classified or sensitive information at the same time preserving software's functionality. We can always treat obfuscation method as a virtual black box or a compiler which could transform the primitive source code to instrumented codes that will behave exactly the same as the original codes. Which means, the obfuscated programs will yield outputs the same with the original outputs given identical inputs.

This Paper is organized as follows: Section 2 addresses current progress in obfuscation research domain, Section 3 states our proposed method Turing Machine in detail, in addition, current progress will be covered as well. Section 4 illustrates experiments we plan to use to evaluate our proposal's performance, Section 5 concludes the whole paper.

2. RELATED WORKS

In Paper "Positive results and techniques for obfuscation"[8], the essence of obfuscation is analyzed and several common methods for software obfuscation were brought about. Generally speaking, many cryptographic applications exist including "Intellectual property" protection which lies in securing secret algorithms and keys, authority and access controlling and private-key and public-key encryption transformation. The authors focused on access control in this paper. This paper gave the first method for program obfuscation which was proved effective. This paper basically is a theoretical research. It didn't involve with any experiments or simulation. Through using Oracle model, the paper proposed a method of reducing obfuscation of one family to obfuscating another. The paper illuminated the future research direction.

Paper "Waermarking, tamper-proofing, and obfuscation - tools for software protection"[9] summarized three types of software attack and corresponding technical defense methods of three main categories including obfuscation. Based on what this paper expressed, obfuscation technology is targeting reverse engineering attack. Through software obfuscation, instrumented software is still functional but unintelligible. Watermarking is another defense category which deals with software privacy probing which could enable identification of a certain assembled software. The last technology mentioned is tamper-proofing which aims at disable the software functionalities in unauthorized copies. [9] is a typical overview journal paper which summarize current research conditions and progresses so that further research could be well-guided. We can literally take this paper as a survey of software security.

Paper "On the Concept of Software Obfuscation in Computer Security"."On the concept of software obfuscation in computer security."[7] proposed that the requirements for obfuscation should not be universal but dependent on a specific program to be obfuscated. This paper mainly talked about obfuscator's security requirements. From their perspective, it makes sense to do varies kinds of obfuscation

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.
†None.

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

with respect to different software security definition. In total, authors of this paper proposed five kinds of model including "black box" and "grey box" which both belongs to total obfuscation, software protection obfuscation, hiding constant obfuscation and predicate obfuscation.

Paper "Indistinguishability Obfuscation for Turing Machines with Unbounded Memory"[6] is a research paper very close to my research topic. The authors of this paper showed how to build indistinguishability obfuscation with modest overhead. The overhead growth was polynomial in terms of security parameter lambda and Turing machine input X. To achieve this goal, they invent a novel primitive which enable the accumulator to own a much larger memory with only a small commitment. The proposed primitive even allow different programs equivalent when they were not at the same stage. It means that this novel primitive is indistinguishability obfuscation friendly. Although this papers' research topic is very similar to mine, the research content was distinct from my Turing obfuscation research. This paper focused on theory proof taking advantage of the Turing machine as a model. However, mine research cares more about Turing encoding and obfuscation implementation and performance evaluation.

Papers above demonstrate theoretical research in obfuscation which is a very crucial part of obfuscation area. In addition, some recent papers published in conferences showed more interests in research on obfuscation implementation methods.

Paper "Translingual Obfuscation"[10] is a paper of such kind. "Translingual" is a new term invented by the authors of this paper. The basic idea behind this term is to convert programs' language from one to another in order to achieve obfuscation effect. Through the "misusing" of the semantics and syntax of a certain programming language, input source program could easily become much more complicated. In particular, this translingual method only transform a portion of source codes which technically made a mix of two or more programming language. In this paper, the authors used Prolog as the target language and C language as the source language. The reason for this Prolog choice may come from its complicated syntax which could make translingual obfuscation more powerful. Prolog's two feature: unification and backtracking were used to make an obfuscation tool called "BABEL". In this way BABEL obfuscation tool could provide effective and stealthy obfuscation. The authors also did lots of experiments to measure the effectiveness of "BABEL". BABEL proved to be superior to common used commodity obfuscation tool in terms of four traditional bench marks used for evaluation obfuscation performance: Potency, Resilience, Cost and Stealth.

In paper "Control flow obfuscation with information flow tracking"[1], the authors proposed a novel application of information flow tracking which has been widely used in securing software execution with only a modest overhead during program execution. This paper took advantage of two key features which are called "the architectural support for automatic propagation of tags" and "violation handling of tag misuses" in information flow tracking area[1]. The novelty in this paper lies in the tag usage. They use the tags during propagation as a flow sensitive predicates in order to hide normal control flow transfers. Which means the real control flow transfer happens at the same time tags are leveraged as predicates for control flow transferred to the violation handler. Further experiments were conducted to test the effectiveness of this method and the experiment results showed that their proposed scheme "BOSH" indeed could obfuscate the whole control flow with minor overhead added.

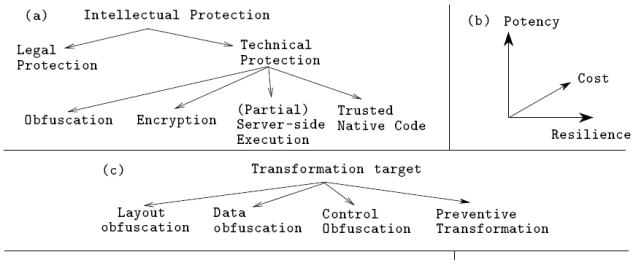

Figure 1: Taxonomy of Intellectual Protection

There are several techniques to protect sensitive information in software including obfuscation, encryption, partial server-side execution and trusted native code[2], which is expressed by figure 1. In this paper, authors argued that automatic code obfuscation is the most promising method with respect to potency, resilience and cost. The main contribution of this paper is the proposed insight that it is acceptable and normal for obfuscated programs to behave differently than the original source codes. Which means that instrumented codes should run slower and be larger. As long as the observable behavior experienced by a user of this original and obfuscated pair is identical, the purpose of obfuscation is achieved.

Paper "Deobfuscation of Virtualization-Obfuscated Software"[3] focused more on de-obfuscating malware. The most common reverse engineer model is to reverse code interpreter first and then based on the accomplishment to figure out program's control flow and algorithm logic. However, this model can not be applied to all cases. This paper's main contribution lied on the proposition of a new approach which identified instructions that could affect those observable behavior of the obfuscated code. Unlike traditional outside-in reversing method, this inside-out approach requires fewer assumptions so the domain of cased eligible for obfuscation would be broadened. Results of deobfuscation malicious code are good.

Book "Reversing: secrets of reverse engineering"[4] is a good tutorial for doing obfuscation. I read chapter one of this book. Chapter one clarified the concept of reverse engineering. More details about reversing were illustrated in the first chapter. Security related reversing, reversing cryptographic algorithms, program binary auditing, reversing development in software and how to achieve interoperability with proprietary software were introduced also.

3. RESEARCH METHOD

In this section, our proposed obfuscation method will be illustrated in detail. Turing Machine related concepts and basics will be covered. Our on going project progress is addressed also.

3.1 Turing Machine

A Turing Machine is an abstract mathematic computation model which is named after British scientist Alan Tur-

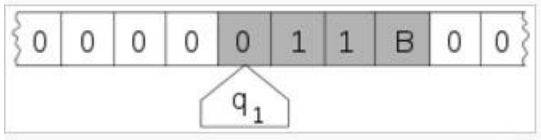

Figure 2: A Turing Machine tape.

Current state	Scanned symbol	Print symbol	Move tape	Final (i.e. next) state	5-tuples
A	0	1	R	B	(A, 0, 1, R, B)
A	1	1	L	C	(A, 1, 1, L, C)
B	0	1	L	A	(B, 0, 1, L, A)
B	1	1	R	B	(B, 1, 1, R, B)
C	0	1	L	B	(C, 0, 1, L, B)
C	1	1	N	H	(C, 1, 1, N, H)

Figure 3: A Turing Machine Transition Table.

ing. This abstract machine model simulates the behavior of a strip of tape under control of a serious of transition rules. There are several components in a Turing Machine such as a header which could read and write on tape(Figure 2). Turing Machine Tape consists of many cells each of which contains a symbol. Cells are beside each other horizontally one by one. The tape is infinite in sense of that it is extendable to left or right. Turing Machine header could read and write on the tape once at a time. After reading and writing, it moves to either left cell or right cell based on the transition rules. Transition table (Figure 3)is a table which holds all the transformation rules. The rules are finite and each of them correspond to a tape and register condition. A transition rule will tell Turing Machine to do the next three things:

- Either erase or write to current cell
- head either mover right or left
- keep the register that same state or assign a new state to it

The state register is used to record the current state of Turing machine. In Figure 3, we could see that any combination of Turing Machine state and tape symbol corresponds to a entry of this transition table which is also represented by a 5 item tuple. A Turing Machine will always work based on the instruction tuple rules to decide which direction to go, what symbol to write and which state to enter . The basic idea for inventing this Turing machine is to convert any logic problem to a Turing Machine mathematical problem. With the help of a Turing Machine, two famous questions could be answered explicitly.

1. Does a machine exist that can figure out that whether a machine conduction on a tape involves with a loop which means the machine can't stop.
2. Is there a machine which could determine whether any arbitrary machine on its tape can eventually print one of the given symbols.

Thus, the computability could be demonstrated by operation a Turing Machine. This is significant because any algorithm's logic could be simulated through operating a Turing Machine despite its extreme simplicity.

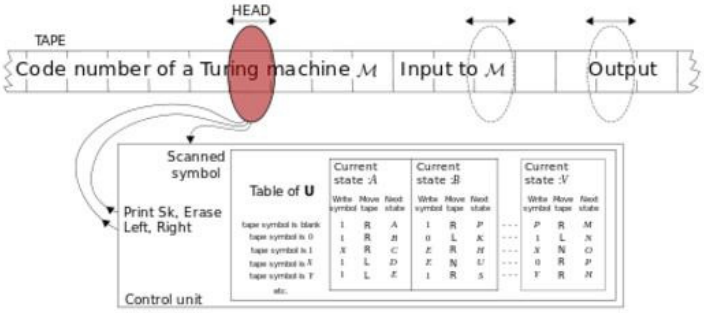

Figure 4: Universal Turing Machine Source:https://en.wikipedia.org/wiki/Universal_Turing_machine

3.2 Universal Turing Machine

Universal Turing Machine(Figure 4) is a special Turing Machine which could simulate multiple arbitrary Turing machine execution. Every Turing machine computes a certain fixed partial computable function from the input strings over its alphabet. In that sense it behaves like a computer with a fixed program[5]. Universal Turing machine could encode any action table in string at the same time constructing a tape which delineates input parameters. Hence, a universal Turing machine could obtain arbitrary Turing machine computation results combining the action table encoding and input table construction. Universal Turing Machine could be taken as an interpreter of a specific Turing Machine.

A Turing Machine takes an input data and yield the output like 'x + 1' after internal running based on its transition table. However, a Universal Turing Machine takes Turing Machine as an special input also. In Figure 4, we could see that a specific Turing Machine implementation is recorded on a tape just like regular input data. The Universal Turing Machine could construct this TM based on the information of this TM. In other words, a Universal Turing Machine could carry any specific Turing Machine and achieve the same functionality of it just like the TM runs given the same data input. In this sense, Universal Turing Machine is universal. In a word, A Universal Turing Machine not only takes data input but also takes Turing Machine as a input.

3.3 Turing Machine Encoding

As all programs' essence is a Turing Machine, every piece of program could be abstracted and transformed to a Turing Machine fulfilling the same functionality. In obfuscation practice, the fact is that we don't need to change every line of source program to do the obfuscation. We are particularly interested in some operators that involve with the control flow graph of the whole program such as '>','<','==', these operators usually change the control flow graph of a certain program. On top of that, arithmetic operators like '+', '-', '*' and '/' are also worth being transformed into Turing Machine in consequence of that they are concrete instructions that build up the whole program. In our research proposal, we plan to extract all these "special" operators from the source codes and use Universal Turing Machine to encode all the predicate codes and arithmetic code part. After encoding, the program will become a mix of Turing implementation codes and original source codes though they are

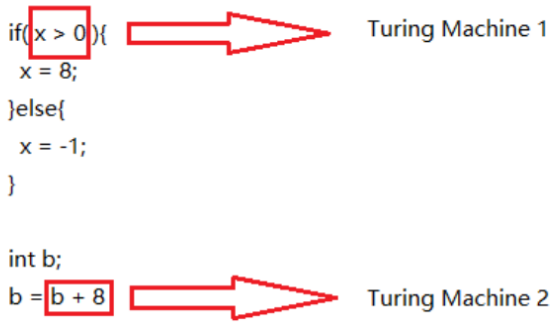

Figure 5: `Instrumented Codes`

both C codes(Figure 5). In this way, we could decide which branch direction to go based on a Turing Machine's final running result in each control flow node. Running a Turing Machine is a prerequisite to enter any program branch.

The goal of obfuscation is to hide a program's internal logic or at least make it complex. It is a novel idea to obfuscate a program through encapsulating predicate lines with Universal Turing Machines. On obfuscation level, we add another layer of abstraction which needs to be parsed when programs are being executed. Internal logic is consequently far more complicated. Hence, to do reverse engineering to a Turing Machine encoded program is hardened to a great extent even impossible.

3.4 Benefits

To our best knowledge, no one has used Turing Machine as an obfuscation model in practice. Some researchers did have prove in theory that it is plausible and achievable to utilize Turing Machine to do software obfuscation, but no one has tried that yet. From my perspective, obfuscated programs would be much harder to be attacked with the help of Turing Machine model, given the complex fact Turing Machine is have to build. It means attackers have to hack each Turing Machine in our instrumented mix codes in order to manipulate the origin source codes. It is highly possible that Turing Obfuscation will yield a much stronger performance.

3.5 Current Progress

To make our research plan more applicable and more detailed, we have divided this project into several sections.

1. Literature review in software obfuscation domain, get to know the most cutting edge obfuscation technology in this field
2. Construct Turing Machine prototype including all components such as tape, transition table etc.
3. Contruct Universal Turing Machine
4. Implement all operators like '+','-' and so on
5. Replace all integer expressions which used these operators with different Universal Turing Machine in source codes
6. Encoding arithmetic expressions in more complex data structures like Abstract Syntax Tree etc.
7. Run a large volume of experiments to test our method's performance.

For now, I have finished to first three steps of the whole process. Turing Machine and Universal Turing Machine was implemented in C language including different components such as tape and transition table. Turing Machine's transition table is implemented by a list consisting of all possible rules. Tape is also a list in which each item node represent a tape slot with a symbol. Arithmetic expression 'x + 1' has been implemented and tested. The rest operators will be implemented in the future.

4. EXPERIMENT DESIGN

There are several common benchmarks used to measure the performance of a obfuscation method: potency,resilience,cost and stealth. We decide to leverage these benchmarks to describe our Turing Machine obfuscation quantitatively. Besides this,in every aspect of evaluation, we need a commonly used obfuscation technology to compare our proposed Turing Obfuscation with. Code Virtualizer(CV) is a commonly used commercial obfuscator. We will compare Turing obfuscation performance with CV obfuscation performance.

4.1 Potency

Basically potency is a index of how powerful a certain obfuscation method is. Some program attributes could indicate this benchmark such as the number of Call Graph Edges(CGF), number of Control Flow Graph(CFG) edges, number of basic blocks in the program, cyclomatic number and knot count. Intuitively, the bigger the number of these attributes is, the more complicated a program is and the stronger a obfuscation method is accordingly. These metrics have been used to measure obfuscation performance in academia. So potency will be measured in terms of these attributes and through the comparison between ours and target obfuscation method.

4.2 Resilience

Resilience is a valuable benchmark that used to indicate whether a obfuscation method is robust enough to resist existent deobfuscation methods. It is straightforward that if a pair of original and obfuscated programs share a large amount of similarities after being processed by a deobfuscation tool, most obfuscation effect is undone by the deobfuscation technique. To our knowledge, comparing deobfuscated obfuscation pair's similarity is a common way to quantify a certain obfuscation technique's resilience. There are not many deobfuscation tools available on market currently. We choose a tool named binary diffing since it is a common tool used in obfuscation academia to test resilience performance.

As we choose similarity as an intermediate index mirroring obfuscation technique resilience, we must define this similarity metric. Generally there are two ways of calculating similarity between a origin version and corresponding obfuscated version. One is syntax based similarity, the other one

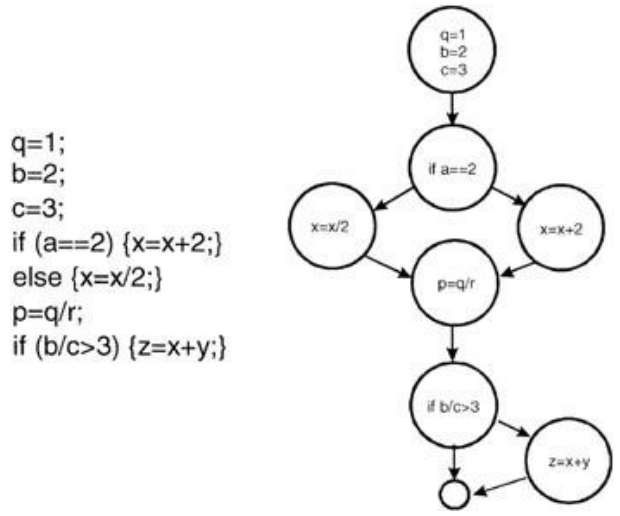

Figure 6: `Control Flow Graph Example. Source:A Practitioners Guide to Software Test Design:Chapter 10`

is semantic based similarity. Syntax based similarity is calculated with the help of the call graphs of a program pair. By intuition, the function that is a isomorphism node and resides in a similar call graph is very likely to be the one serves the same computational logic. However, semantic based similarity focus more on precondition and postcondition of the execution of a certain block of codes. If two blocks of codes produced a same result state give the same precondition, we could say that this pair of codes are similar from each other.

4.3 Cost

We all know that obfuscation would by all means induce some extra overhead, while to what extent the overhead is added is crucial for running a obfuscated program. There is a threshold that differentiate whether the obfuscation cost is acceptable. We would never adopt an extremely slow technique after all. Traditionally, cost means two things, memory and cpu. We intend to run Turing obfuscated program 10 times in order to get an average measurement of CPU and memory consumption. Compare this result with 10 times average CPU and memory consumption when running the original program, we could assess the cost performance. Similarly, we will choose code virtualizer as a compare group in experiment. CPU consumption and memory usage will be measured during the whole running process of original program and obfuscated version.

4.4 Stealth

Obfuscated program should resemble the origin version program. Stealth is a practical metric as we wouldn't like our instrumented programs appear evidently distinguish but try to be as inconspicuous as possible. We plan to take advantage of assembly instruction distribution to describe whether Turing Machine obfuscation is stealthy. If the distribution of each main instruction such as 'mov','call','cmp' is statistically similar which means distribution level fall into the normal standard deviation of mean percentage, we say that the obfuscated program is similar with the original and remains stealthy. If not, to some extent the obfuscated codes are much different from the source codes, it doesn't hide well and is not stealthy.

4.5 Expected Results

Usually it is not possible to improve all the benchmarks through a new technology, it is like a trade off in most research cases. We are more interested in improving the Potency and Resilience with our method since it would indicate how complicated our instrumented program is. This is the essence of an obfuscation model. Specifically, we expect more call graph edges, more CFG edges, more basic blocks and more cyclomatic number in obfuscated programs using our method. In this way we could say that the potency of obfuscation is proved. In addition, we expect a low similarity between original C source codes and obfuscated source code processed by our proposed method. That means deobfuscation tool can not undo our obfuscation effect induced by our method. The more similar original codes is with processed codes, the less resilient a certain method is. Focusing on improving these two aspects, we think that some minor increase in resource cost and downgrade of stealth is still acceptable.

5. CONCLUSIONS

Software obfuscation is a very crucial in such a age facing more and more online attacks.This paper introduce our research project Turing Machine Obfuscation. We proposed a novel idea that which utilizes Turing Machine encoding as the obfuscation method. Through transforming the target instructions which are mainly predicates and arithmetic operations into the execution of corresponding Turing Machine, internal control flow and computational logic are complicated to a large extent. From our perspective it is a way worth researching.Besides, this paper summarised my current progress in the project and list the future research steps as a guideline. We plan to evaluate this novel technique in terms of four traditional common metrics that are potency,resilience, cost and stealth respectively. With this new obfuscation method, we believe the experiments may yield good results.

6. REFERENCES

[1] e. a. Chen, Haibo. Control flow obfuscation with information flow tracking. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM*, 2009.

[2] C. T. Collberg, Christian and D. Low. A taxonomy of obfuscating transformations. Department of Computer Science, The University of Auckland, New Zealand, 1997.

[3] G. L. Coogan, Kevin and S. Debray. Deobfuscation of virtualization-obfuscated software: a semantics-based approach. In *Proceedings of the 18th ACM conference on Computer and communications security. ACM*, 2011.

[4] E. Eilam. Reversing: secrets of reverse engineering. In *"John Wiley and Sons"*, 2011.

[5] https://en.wikipedia.org/wiki/Universal_Turing_machine.

[6] A. B. L. Koppula, Venkata and B. Waters. Indistinguishability obfuscation for turing machines

with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, 2015.

[7] Kuzurin and Nikolay. "on the concept of software obfuscation in computer security.". In *Information Security*, pages 281–298. Springer Berlin Heidelberg, 2007.

[8] B. Lynn, M. Prabhakaran, and A. Sahai. Positive results and techniques for obfuscation. In *Advances in Cryptology-EUROCRYPT 2004*, pages 20–39. Springer, 2004.

[9] C. C. S and Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. In *Software Engineering, IEEE Transactions*, pages 735–746, 2002.

[10] S. W. e. a. Wang Pei. Translingual obfuscation. In *arXiv preprint arXiv:1601.00763*, 2016.