"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XIX – 2016 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania // The journal is indexed in: PROQUEST / DOAJ / DRJI / JOURNAL INDEX / I2OR / SCIENCE LIBRARY INDEX / Google Scholar / Crossref / Academic Keys / ROAD Open Access / OAJI / Academic Resources / Scientific Indexing Services / SCIPIO

SOFTWARE SECURITY TECHNIQUES: RISKS AND CHALLENGES

Marius Iulian MIHAILESCU[1]
Stefania Loredana NITA[2]
Marian Dorin PIRLOAGA[3]
[1]Department of IT&C, LUMINA – The University of South-East Europe
[2]Department of Integrated Systems, Institute of Computers
[3]Military Technical Academy

Abstract: *Because of the increasing number of applications that are working on-line, software security has become an important aspect for software development process. The paper will present the main mechanisms and features on which we have to stop when we are designing and implementing a software application, such as sensitive information, execution of the program, and different ways of analyzing static and dynamic code. We will explain two attacks techniques (analysis and tampering) that could occur on the program and we will demonstrate how we can exploit some vulnerable points of access in the software application. Based on the two types of attacks we will discuss about obfuscation techniques and perturbated functions as a new approach to obfuscation and diversity.*

Keywords: *software security, obfuscation, perturbated functions, client-server, attacks*

Introduction

Nowadays we are facing with a real challenge regarding the security of software applications within a company or a personal computer. When we are talking about security for a software application we have to concentrate on four general questions: (1) where the application will be installed (local network, business computer, personal computer, cloud computing environment)?; (2) who will have access to the application (types of users)?; (3) how the application will be accessed (authentication methods)?; (4) which are the security techniques used and where in the source code of the application have been implemented and how?. Behind of this process everything its quite complicated and the goal of this paper is to present a framework that need to be applied when an application will be developed and deployed within a business of personal environment.

Many companies are developing software applications without a strategy for assuring and finding the right security techniques for the applications during the development process (e.g. protecting the code against different attacks as we will discuss later in this paper). Such strategy could be Software Security Assurance (SSA), which is known as the technique included in the development phase of the software applications. The SSA is operating at a level of security that is very consistent with the potential threats that could come out from the loss, inaccuracy, alteration, unavailability, or misuse of the data and resources that it uses, controls, and protects.

Naturally speaking, an adequate security is necessary in this mixed and heterogeneous environment. As we can point out, a software contains secret, confidential or sensitive information. Let's take for example, the medical files or credit card numbers. In order to protect this data, there exist encryption and authentication algorithms [2].

The paper will discuss about software obfuscation and it will present some of the most common techniques used in software development process in order to protect the sensitive code and not only.

The paper is structured in seven sections (excluding the introduction and conclusions sections) as it follows: (Section 2) Obfuscation; (Section 3) Software Protection Problems; (Section 4) Attacks on Software; (Section 5) Code Transformations; (Section6) The Proposed Framework.

Obfuscation

PC programs speaks to the most complex questions that have been developed by people. Notwithstanding understanding a little program, for example, a 10-line project, for example, the one displayed in Fig. 1 can be amazingly troublesome. The multifaceted nature of projects had turned into the bane (and extremely well the shelter) of the product business, and the appeasement process have turned into the fundamental objective of industry and scholarly. Beginning from this, we can discover a few angles not all that shockingly for both, theoreticians and professionals which have been attempting to "tackle this unpredictability for good" and use it some way or another to ensure touchy data, and obviously calculation. This is known as programming confusion, and the exchange from our article will associate with this idea.

Any cryptographic instrument, for instance encryption or confirmation can be considered as

DOI: 10.21279/1454-864X-16-I1-077

acknowledging many-sided quality security, yet with programming jumbling, the general population begin going for something more yearning, suppose as a method for changing subjective projects into something like muddled.

As per [21] the exploration on muddling is in an ,,embryonic stage". This announcement depends on the way that there are no down to earth effectiveness verification, yet we have just hypothetical evidences which are arranged extremely distant from the practice.

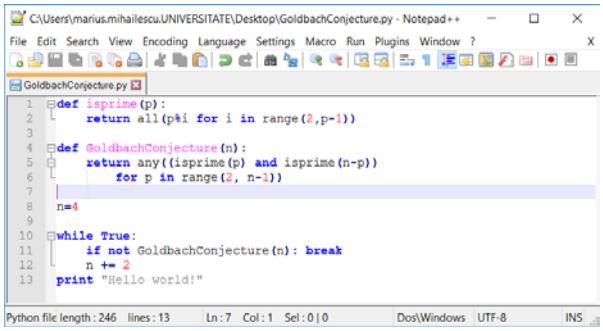

Fig. 1. A simple program that could be obfuscated in Python

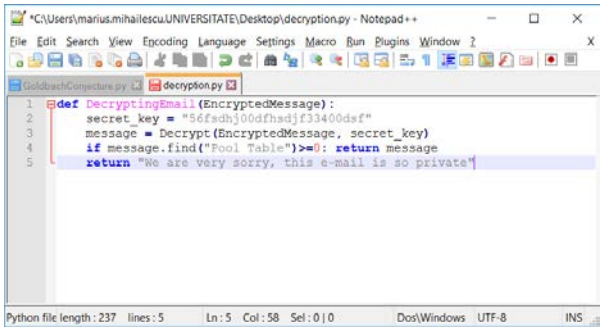

Fig. 2. A simple program used for code encryption

Software Protection Problems

We take a gander at programming assurance from a designing perspective. Those procedures does not fit into white-box model, as we have depicted in Section 1.

In this section we will concentrate on a few arrangements with respect to customer server procedures, methods to obstruct programming investigation, Collberg's obscurity changes, code changes, and an exceptional discourse will be on confusion measurements.

One of the goal is to give a cutting edge for programming assurance methods. As a short audit of the fundamental commitments, we can express: a review of programming insurance systems, an examination of strategies judged on their capability to secure against investigation and altering assaults, both static and element.

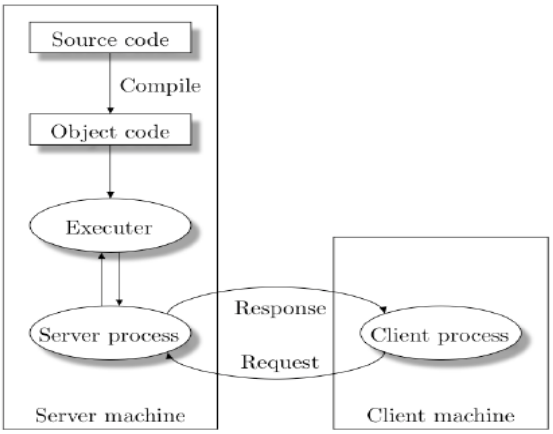

Fig. 3. The client-server model only distributes access to services but not to the code, which is running at the server side [1]

Client-Server Solutions

A standout amongst the latest systems keeping in mind the end goal to ensure the basic programming was to run it at the proprietor side rather than the client side. This procedure or system is referred to as programming as an administration. For this situation, the basic programming was not appropriated to untrusted has, but rather it had been kept up on a very much ensured server. The assurance of the server is made from system, equipment, and programming security. More often than not, the code itself is not secured by some other procedures. As per this setup, the administrations are dispersed and not the product itself, as we can see in Figure 1. Source code and the executable code dependably will be on the server side. In the event that we are an assailant, the server will be seen as a black box which can't be gotten to by sending reactions of solicitation and getting.

The administration is currently dispersed over the customers and the server. There are some focal points, for example, the lessened size of the server, just additional overhead is required to keep up the correspondence between the customer and the server. This perspective will raise an alert cautioning on the way that this procedure of correspondence will speak to the fundamental issue. At a fast look, the said model will empty the server, yet when we look practically speaking the customer and the server require an extremely escalated correspondence so that the transfer speed will turn into a bottleneck.

DOI: 10.21279/1454-864X-16-I1-077

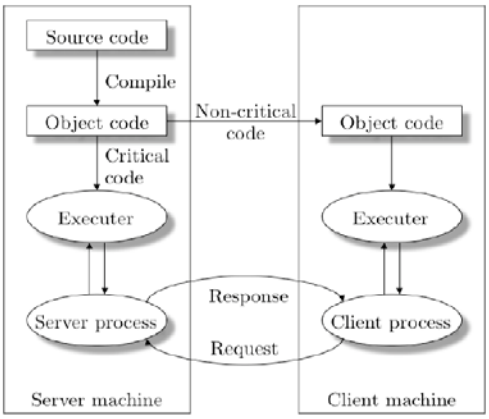

Fig. 4. The partial client-server model splits code into a critical and noncritical part: the critical part is run at the server side; the non-critical part is run at the client side [1]

Techniques to Thwart Software Analysis

Underneath, we will attempt to get a handle on various strategies that can make an insurance against investigation. The point of the most procedures that are available today is to ensure against figuring out [12], statically or powerfully [13] [14].

A portion of the strategies said above can change the code when the application is disconnected from the net or amid the runtime process. For this situation, both classes will increase current standards for an assailant that desires to make an appropriate examination, and obviously, it will have the capacity to postpone an altering assault also.

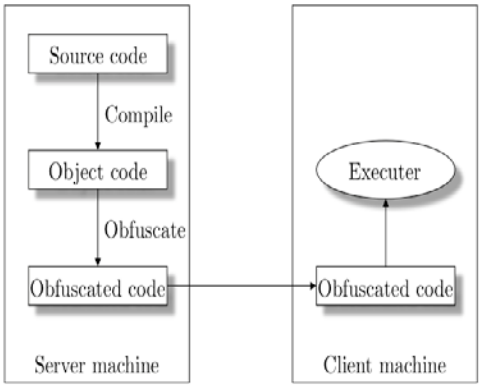

Fig. 3. Obfuscation Model [1]

3.1.1. Collberg's Obfuscation Method for Transformation

Object-situated writing computer programs is connected all over the place since it offers diverse focal points to peruse, adjust and amplify your code.

Programming in modules will leave diverse tracks into the executable and this will abuse these imprints and follows keeping in mind the end goal to remake the first source code [15]. As a short history, when Java bytecode get to be defenseless at decompilation [16], yielding the first source code, the analysts had begun to research the procedures with a specific end goal to secure the first source code [17] [18].

One fascinating view point, is the real trick proposed by Collberg's [19], where he characterizes jumbling as a procedure of change that endeavors to change a project into a something comparative which is extremely hard to figure out. The examination in view of code obscurity applies one or more code changes stages which will make the code more impervious to investigation and altering. There is one single hindrance which comprise in holding its usefulness. For this situation, our code can be circulated over various untrusted has without expecting any sort of dangers that could be figured out (see Figure 3).

As indicated by Collberg et al [20], we need to concentrate on four fundamental classes of code muddling changes:

- Lexical change;
- Control stream changes;
- Data stream changes;
- Preventive changes;

For more insights about these classes of code obscurity changes, you may discover here [1], beginning with page 30.

Attacks on Software

There are two fundamental classifications of assaults on programming that could occur, for example, static assaults and element assaults.

The primary commitments of this segment is to show an assurance plan that will augments the control stream chart leveling which is more grounded against static control stream investigation, three models that guide our plan onto application situations, and some assaults to delineate the quality of our plan.

Static Attacks

Static investigation which alludes to examinations which don't include execution of the real code. Compilers depend on static examination methods with a specific end goal to streamline code. For instance, consistent engendering and liveness examination. Figuring out is utilized with a malevolent intention.

Somebody has something and needs to comprehend what it does, how it does this, and so on. A figure out commonly begins examining an item, by dismantling it, and after that tries to comprehend it a little bit at a time, forming parts, discovering designs, and so forth. In programming, a fundamentally the same procedure happens. Initial a double record is dismantled. As a second step, the figure out might

DOI: 10.21279/1454-864X-16-I1-077

pick to decompile the dismantled code into source code. Lastly, he will assess the source code. Note that the figure out can likewise essentially run the code he acquired.

There are to systems which can be connected on the code: dismantling and decompilation.

Disassembing

At the point when figuring out a double record, an initial step comprises of dismantling the program into a human justifiable configuration. The dismantling step is the backwards of the gathering stage in compilers. It makes an interpretation of paired code into get together directions that accommodate a specific CPU engineering. While this is a static system, it is not an immaculate skill, as useful disassemblers need to depend on suppositions [1].

Decompilation

Decompilation is really a discretionary system that the figure out can apply. On the off chance that an aggressor needs to comprehend a whole program, he may be faced with a huge number of lines of gathering code. A decompiler essentially searches for examples that can be deciphered into source code. As abnormal state code is wealthier and more conservative, it is frequently less demanding to comprehend [1].

How to protect against static analysis

While encryption regularly is introduced as the way to ensure programming statically, it really moves the issue, simple to cryptography where mystery of a message is moved to mystery of a key. In an encoded executable record, unique system code is scrambled, and a decoding routine is added to the first program. Consequently, code encryption is a type of self-changing code [23]. Really, the whole program is dealt with as information, while the decoding routine remains code. In the event that the last is anything but difficult to examine, one can "break" the unscrambling schedule, and decode the project. Consecutive steps, for example, disassemblation and decompilation permit to figure out it, as though it were never secured.

Besides, not all designs at present bolster is self-adjusting code.

Some working frameworks implement a Q⊕R strategy as a system to make the misuse of security vulnerabilities more troublesome. This implies a memory page is either Writable (information) or executable (code), yet not both. Scrambled code should strife with infection scanners because of its suspicious conduct (malware additionally utilizes self-decoding code) or because of false-positive marks matches, i.e. bit designs that infection scanners check for. A workaround for this impediment is the utilization of a virtual machine [22]. On the off chance that code is arranged in the nick of time, the virtual machine can run it. In the event that the virtual machine is mystery, and the byte code is encoded, one needs to assault the virtual machine first.

Dynamic Attacks

On the off chance that a contender succeeds in extricating and reusing an exclusive calculation, the results might be noteworthy. Besides, mystery keys, classified information, or security related code are regularly not expected to be dissected, separated, stolen, or defiled. Indeed, even within the sight of legitimate defends, for example, licensing and cybercrime laws, figuring out remains a significant danger to programming engineers and security specialists.

By and large, the product is figured out, as well as messed around with, as exemplified by the multiplication of breaks for gaming programming and DRM frameworks. In a branch sticking assault, an assailant replaces a contingent hop by an unequivocal one, compelling a particular branch to be taken notwithstanding when it shouldn't under the anticipated conditions. Such assaults could majorly affect applications, for example, authorizing, DRM, charging, and voting.

Code Encryption

The objective of encryption is to shroud data. Initially, it was connected inside the setting of correspondence, yet has turned into a procedure to secure an expansive scope of basic information, either for transient transmission or long haul stockpiling.

All the more as of late, business instruments for programming insurance have ended up accessible. These devices need to shield against assailants who can execute the product on an open design and in this manner, yet in a roundabout way, have entry to all the data required for execution.

This area gives a diagram of runtime code decoding and encryption.

One can likewise allude to this as a particular type of self-adjusting or self-producing code.

Encryption guarantees the privacy of information. With regards to paired code, this strategy for the most part ensures against static examination and altering. For instance, encryption methods are utilized by polymorphic infections and polymorphic shell code. Along these lines, they can sidestep interruption.

Bulk Decryption

We allude to the strategy of decoding the whole program immediately as mass unscrambling. The decoding routine is generally added to the scrambled body and set as the section purpose of the system. At run time this routine unscrambles the body and after that exchanges control to it.

DOI: 10.21279/1454-864X-16-I1-077

The decoding routine can either counsel an inserted key or get one powerfully (e.g. from client information or from the working framework). The fundamental point of preference of such a system is, to the point that the length of the project is scrambled, its internals are covered up and along these lines ensured against static investigation.

Another point of preference is that the encoded body makes it hard for an aggressor to statically change bits meaningfully. Changing a solitary piece will bring about one or more piece flips in the unscrambled code (contingent upon the blunder proliferation of the encryption plan) and hence one or more adjusted directions, which may prompt project crashes or other unintended conduct because of the weakness of parallel code.

Notwithstanding, as all code is decoded at the same time, an assailant can essentially sit tight for the unscrambling to happen before dumping the procedure picture to plate.

On-Demand Decryption

Rather than mass decoding, where the whole program is unscrambled without a moment's delay, one could build granularity and unscramble little parts at the point in time when they are really required. When they are no more required, they alternatively can be re-encoded. This method is for instance connected by Shiva, a paired encryptor that utilizations muddling, hostile to investigating systems, and multi-layer encryption to secure x86 doubles utilizing the Mythical person position.

On-interest unscrambling defeats the shortcomings of uncovering all code free without a moment's delay as it offers the likelihood to decode just the fundamental parts, rather than the entire body. The hindrance is an expansion in overhead because of numerous calls to the unscrambling and encryption schedules.

Attacks and Improvement

Our gatekeepers, which alter code contingent upon other code, offer a few favorable circumstances over the product protects proposed by Chang and Atallah [24] and the from introduced by Horne et al. [26]. A review is given underneath:

Classification. To start with, all capacities are scrambled statically, either by mass or by on-interest encryption. An aggressor breaking down code statically is compelled to first infer all dynamic unscrambling keys and after that decode the code. Besides, the length of code remains scrambled in memory it is ensured against memory dump assaults. With a decent plan it is plausible to guarantee just an insignificant number of code pieces are available in memory in decoded structure. Thus, an aggressor dumping memory would just have the capacity to assess works part of the call stack. Exchanging off security for execution, utilizing the hotness heuristic, chooses more capacities for mass encryption, consequently making them helpless to element examination.

Alter resistance. Together with a decent reliance plot, our watchmen offer assurance against endeavors to adjust the project code. On the off chance that a capacity is messed with statically or even progressively, the system will produce debased code at a later stage and in this way it will in the end crash because of illicit guidelines or yield questionable results. Besides, if the adjustment produces executable code, mistakes will in any case show up in different capacities. An assailant utilizing a debugger to step-follow through the system, may fall flat too. For instance, the Unix debugger gdb [27] utilizes programming breakpoints. These product breakpoints adjust the stacked code in memory. In the event that the comparing code is either hashed (to determine a decoding key) or unscrambled, this will incite flaws.

Imperviousness to an equipment helped circumvention assault. An assault, proposed by van Oorschot et al. [28], misuses contrasts between information peruses and direction brings to sidestep self-check summing code. The assault comprises of copying every memory page, one page containing the first code, while another contains altered code. A changed bit captures each information read and diverts it to the page containing the first code, while the code brought for execution is the adjusted one. Nonetheless, later work of Giffin et al. [25] represents that self-altering code can identify such an assault and along these lines ensure against it. As our work concentrates on self-encoding code, a kind of self-altering code, the discovery system of Giffin et al. likewise applies to our procedure.

Code Transformations

Business muddling programs frequently just scramble identifier names and evacuate excess data, for example, investigate data, in code. This is entirely unimportant, however jumbling offers significantly more potential outcomes. A decent confusion exists out of one or more program changes that change a project's control and information stream in a way it gets to be harder to investigations and figure out. However, the main limitation for these changes is safeguarding the usefulness of the first program. Consequently, obscurity is an accumulation of numerous systems that are helpful for project change, confusion or randomization.

Besides, the greater part of these code changes are not one way and it is difficult to choose where

DOI: 10.21279/1454-864X-16-I1-077

to utilize which changes. Hence, a few parameters measure the nature of a change reasonable for code obscurity:

- The fundamental confinement remains safeguarding the system usefulness.
- The fundamental objective of code changes is maximal confusion of the first program.
- A change needs as much imperviousness to robotized assaults.
- A change should be as stealthy as could reasonably be expected, too for static as dynamic investigation methods.
- Increase in code size and execution time should be minimized.

Regardless these systems don't promise waterproof security, a blend of a few change procedures can prompt adequate handy assurance against figuring out and altering assaults.

The Proposed Framework

In this section we will propose a framework that is important to be connected in two phases of the product improvement stages, the main stage is investigation and the second one is outline. On the off chance that the structure is connected with accomplishment over the stages specified beneath, then the usage stage will be done much less demanding and the dangers to make security gaps and breaks will be minimized.

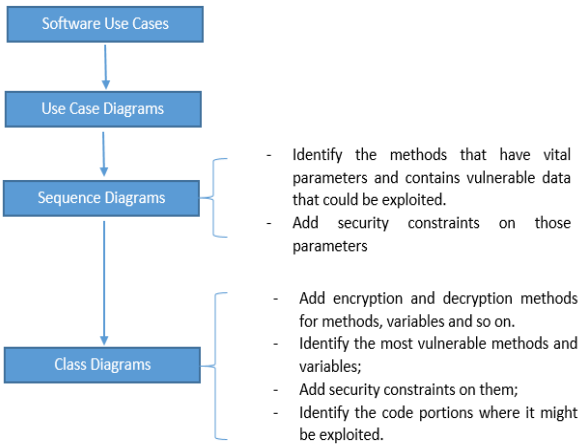

Fig.4. A basic framework for software security

Our system depends on four basic steps. In the event that the above steps (see Figure 4) are tailed, we can maintain a strategic distance from a considerable measure of bugs and security blemishes.

It must be clear from the earliest starting point the goal of the application, where it must be introduced, what are the touchy information, the client validation prepare, the encryption calculations, how the encryption techniques are utilized etc.

Every stride speaks to a dive deep into the examination of the product.

The structure of the system is as per the following:

- **First step – Software use cases**. In this phase the software developer will take out the main user events and is trying to characterize it in such waythat he will be able to identify some preliminary sensitive data.
- **Second step – Use Case Diagram.** In this phase the software developer will see how the events will interact between them. Which are the users, what roles they have. This is a good step because it has a full overview on the entire system and boundaries.
- **Third parameter – Sequence Diagram.**Here the software developer and analyst will have an overview over the methods initiations and calls together with parameters. Here the methods that have vital parameters and contains vulnerable data will be treated with a maximum attention. In this phase, will introduce the security constraints which can be added on methods or variables.
- **Forth parameter – Class Diagram.**The entire structure of the application can be seen on this diagram. Nothing will escape from this diagram. If this diagram is made in a professional way, then the application will be developed in the same way. In this step the software analyst and developer could add special encryption and decryption methods, will identify the main classes and interfaces that can be exploited, different portions of code will be identified and different security levels will be attached. The security levels will indicate how vulnerable the code is.

This framework is a little time consuming, but it is worth it. We can keep the track of everything that show and give us the possibility to issue a security hole into the application.

The next step is to apply this framework automatically. In order to achieve this, we will implement as an add-on or plugin for NetBeans IDE and Microsoft Visual Studio 2015. It will be available to download it from NuGET at the end of 2016.

DOI: 10.21279/1454-864X-16-I1-077

CONCLUSIONS

In the end, we will like to mention that our research for this paper was a real challenge especially when we have tried to cover the most important aspects about software security techniques, and to point out the main risks and advantages.

The main goal of the paper was achieved, but there are other many things that need to be mentioned and just a simple article is not enough.

We have proposed a framework which is required to follow when a new software is designed and ready for the implementation phase.

BIBLIOGRAPHY

[1]. Jan Cappaert, Code Obfuscation Techniques for Software Protection, Dissertation presented n partial fulfillment of the requirements for the degree of Doctor in Engineering, Arenberg Doctoral School of Science, Engineering & Technology, Faculty of Engineering, Department of Electrical Engineering (ESAT), KatholiekeUniversiteit Leuven, 2012.

[2]. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of AppliedCryptography. CRC Press, 1996.

[3]. Java 2 platform security architecture.http://docs.oracle.com/javase/1.4.2/docs/guide/security/spec/securityspec.doc.html (consulted on April 22th, 2016).

[4]. The LLVM compiler infrastructure.http://llvm.org (consulted on February 10th, 2012).

[5]. The International Obfuscated C Code Contest.http://www.ioccc.org/ (consulted on April 22th, 2016).

[6]. Trusted computing group.http://www.trustedcomputinggroup.org/ (consulted on April 22th, 2016).

[7]. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Princiles, Techniques,and Tools*. Addison-Wesley, 1986.

[8]. B. Anckaert. *Diversity for Software Protection*. PhD thesis, GhentUniversity, 2008.

[9]. B. Anckaert, B. De Sutter, D. Chanet, and K. De Bosschere.Steganography for executables and code transformation signatures. InC. Park and S. Chee, editors, *ICISC*, volume 3506 of *Lecture Notes inComputer Science*, pages 425–439. Springer, 2004.

[10]. B. Anckaert, M. H. Jakubowski, and R. Venkatesan. Proteus:virtualization for diversified tamper-resistance. In M. Yung, K. Kurosawa,and R. Safavi-Naini, editors, *Digital Rights Management Workshop*, pages47–58. ACM, 2006.

[11]. B. Anckaert, M. H. Jakubowski, R. Venkatesan, and C. W. Saw. Runtimeprotection via dataflow flattening. In R. Falk, W. Goudalo, E. Y. Chen,R. Savola, and M. Popescu, editors, *SECURWARE*, pages 242–248. IEEEComputer Society, 2009.

[12]. Reverse Engineering, https://en.wikipedia.org/wiki/Reverse_engineering

[13]. Static Program Analysis, https://en.wikipedia.org/wiki/Static_program_analysis

Dynamically vs. Statically, http://reverseengineering.stackexchange.com/questions/11512/dynamically-vstatically-linked

[15]. C. Cifuentes and K. Gough. Decompiling of binary programs. *Software – Practice & Experience*, 25(7):811 829, 1995.

[16]. T. A. Proebsting and S. A. Watterson. Krakatoa: Decompilation in Java(Does bytecode reveal source?). In *COOTS*, pages 185–198. USENIX,1997.

[17]. C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscatingtransformations. Technical Report #148, Department of ComputerScience, The University of Auckland, 1997.
http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow97a/A4.pdf (consulted on February 10th,2012).

[18]. D. Low. Java control flow obfuscation. Master's thesis, University ofAuckland, New Zealand, 1998.

[19]. C. S. Collberg and C. D. Thomborson. Watermarking, tamper-proofing,and obfuscation - tools for software protection, volume 28, pages 735–746, 2002.

[20]. C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscatingtransformations. Technical Report #148, Department of ComputerScience, The University of Auckland, 1997.http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow97a/A4.pdf (consulted on April 25th, 2016).

[21]. Boaz Barak, Hopes, Fears, and Software Obfuscation, COMMUNICATIONS OF THE ACM, vol. 59, no. 3, March 2016.

[22]. T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*.Addison-Wesley, 1999.

[23]. N. Mavrogiannopoulos, N. Kisserli, and B. Preneel. A taxonomy of selfmodifyingcode for obfuscation. *Computers & Security*, 30(8):679–691,2011.

[24]. H. Chang and M. J. Atallah. Protecting software code by guards. InT. Sander, editor, *Digital Rights Management Workshop*, volume 2320 of*Lecture Notes in Computer Science*, pages 160–175. Springer, 2001.

[25]. J. T. Giffin, M. Christodorescu, and L. Kruger. Strengthening software self-checksumming via self-modifying code. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSA05)*, pages 23–32. IEEE Computer Society, 2005.

[26]. B. G. Horne, L. R. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic self-checking techniques for improved

DOI: 10.21279/1454-864X-16-I1-077

tamper resistance. In T. Sander, editor, *Digital Rights Management Workshop*, volume 2320 of *Lecture Notes in Computer Science*, pages 141–159. Springer, 2001.
[27]. R. Stallman, R. Pesch, and S. Shebs. Debugging with gdb: The GNU source-level debugger, 2010.
[28]. P. C. van Oorschot, A. Somayaji, and G. Wurster. Hardware-assistedcircumvention of self-hashing software tamper resistance. *IEEE Trans.Dependable Sec. Comput.*, 2(2):82–92, 2005.

DOI: 10.21279/1454-864X-16-I1-077