On the Need for More Human Studies to Assess Software Protection

Mariano Ceccato
Fondazione Bruno Kessler, Trento, Italy. Email: ceccato@fbk.eu

***Abstract*—Programs often run under strict usage conditions (e.g., license restrictions) that could be broken in case of code tampering. Possible attacks include malicious reverse engineering, tampering using static, dynamic and hybrid techniques. Many code protection techniques (e.g., code obfuscation) have been proposed to mitigate the problem of attacks to software integrity, by turning code resilient to attacks or just more difficult to understand and, consequently, to attack.**

Effectiveness of software protection in limiting or retarding attacks is often assessed by using various code metrics. However, metrics alone give a limited (and potentially biased) quantification of the level of protection. Human studies are required to validate metrics and to objectively quantify how effective is code protection in blocking malicious tampering. Human studies would shown if metrics approximate the actual effort required by an attacker break protections. However, these studies are very expensive and time consuming. The contribution of the whole research community is required to achieve this demanding objective.

I. Introduction

With sufficient effort and resources, most software protection techniques can be defeated under the man-in-the-end (MATE) attack model. In fact, the information needed to break a software system is often present, possibly in obfuscated form, in the executable binary or bytecode. Hence, they can be accessed and controlled by the attacker. Assessing software protections means to estimate the extra delay an attacker would incur due to a particular protection technique used on a given application.

Most of the existing assessments of software protection techniques (in particular, for code obfuscation) are based on metrics, estimating the increased code complexity, or the increased difficulty of static code analysis. Only a few works are based on attacks performed by human subjects, but very few of them applied rigorous approaches, such as those available from the area of empirical software engineering.

II. Assessment Based on Metrics

The evaluation of the increased strength introduced by obfuscation techniques has been mainly addressed by using code metrics. Even if metrics are based on reasonable assumptions about the expected problems that an attacker would face to defeat the code protections, they just estimate and approximate a specific level of security that the underlying application is supposed to receive.

Collberg *et al.* [6] proposed the use of complexity measures in code obfuscation tools to help developers choose among different obfuscation transformations. A high-level approach has been proposed by Collberg et al. [7] when they defined the concept of *potency* of an obfuscation as the ratio between the complexity (measured with any metric) of the obfuscated code and the complexity of the original source code, and the concept of *resilience*, i.e. how difficult it is to automatically de-obfuscate the protected code.

Karnick *et al.* [10] defined more precise metrics for potency (combining nesting, control-flow and variable complexities), resilience (as the number of errors generated when decompiling the obfuscated code) and cost (as an increment of memory usage). Heffner and Collberg [8] used metrics for obfuscation potency and performance degradation as they aimed at finding the optimal sequence of obfuscations to be applied to different parts of the code in order to maximize complexity and reduce performance overhead. With a similar goal, Jakubowski *et al.* [9] presented a framework for iteratively combining and applying protection primitives to code; they also used code size, cyclomatic number and knot count metrics to evaluate the code complexity.

Many authors have chosen just a few specific metrics, under the assumption that these are good indicators of the software complexity and of the task difficulty for attackers trying to break the code. Anckaert *et al.* [1] attempted to quantify and compare the level of protection of different obfuscation techniques, with metrics based on *code*, *control flow*, *data* and *data flow*, without however performing any validation on the proposed metrics. Linn *et al.* [11] define the *confusion factor* as the percentage of assembly instructions in the binary code that cannot be correctly disassembled by the disassembler, assuming that the difficulty of static code analysis will increase with this metrics, even if it strongly depends on the disassembly tools and algorithms used.

Tamada et al. [13] have proposed a mental simulation model to evaluate program obfuscation. The mental model simulates the short term memory of humans as a FIFO queue of limited, fixed size. Then, the authors compute six metrics that account for the difficulty possibly encountered by humans understanding the program, in accordance with the simulation model. They show that the values of such metrics increase – hence making program understanding more difficult – when a number of well-known obfuscation techniques are applied to the program to be protected.

Probably the most comprehensive analysis is represented by the comparison conducted by Ceccato *et al.* [2]. This study considered many different applications from different domains, consisting of more than 4 millions lines of code.

Forty four different obfuscations have been applied to the code. The effects of each obfuscation have been quantified separately as the changes occurred to the code according to 10 metrics, including Chidamber and Kemerer's modularity, (cyclomatic) complexity and size (lines of code). In order to provide reliable results, a statistically sound evaluation has been conducted.

III. Assessment Based on Human Studies

Empirical software engineering is devoted to the design and execution of controlled experiments to study how developers change their productivity while working with alternative tools/approaches. Empirical software engineering requires several steps for carefully defining the experimental environment, such as stating the experimental hypotheses, defining relevant variables to measure, preparing attack tasks to be executed by the participants, profiling the participants, analyzing the data with proper statistics and elaborating the threats to the observation validity.

Despite the benefits of experimental investigation, in the security literature only a few works are based on attacks performed by human participants on binary code [12] and even fewer works [3], [5] are based on sound empirical approaches, because the latter are expensive to conduct and time consuming. As an example, the comparison of just two obfuscation techniques with the involvement of humans playing the role of attackers took a quite long time to be concluded [3].

Sutherland *et al.* [12] conducted an experimental study on the complexity of binary reverse engineering. The authors of this study asked a group of 10 students (of heterogeneous level of experience) to perform static analysis, dynamic analysis and change tasks on several C (compiled) programs. They found that the participants' ability was significantly correlated with the success of threverse engineering tasks they were asked to perform.

Our initial investigation on this topic is described in some works [4], [5], [3] devoted to present the design, planning and results of a series of experiments devoted to compare the level of protection offered by two of the most common code obfuscation techniques, *identifier renaming* and *opaque predicates*. This study compared, by means of statistical tests and effect size measures, the capability and efficiency of participants in performing attack tasks on clear and obfuscated code. The study was able to quantify the increased effort necessary to understand and attack an obfuscated program, with respect to the effort necessary for the non-obfuscated one.

Acknowledge

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 609734.

References

[1] B. Anckaert, M. Madou, B. D. Sutter, B. D. Bus, K. D. Bosschere, and B. Preneel. Program obfuscation: a quantitative approach. In *QoP '07: Proc. of the 2007 ACM Workshop on Quality of protection*, pages 15–20, New York, NY, USA, 2007. ACM.

[2] M. Ceccato, A. Capiluppi, P. Falcarin, and C. Boldyreff. A large study on the effect of code obfuscation on the quality of java code. *Empirical Software Engineering*, page (to appear), 2014.

[3] M. Ceccato, M. Di Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19:1040–1074, 2014.

[4] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. Towards experimental evaluation of code obfuscation techniques. In *Proceedings of the 4th ACM workshop on Quality of protection*, QoP '08, pages 39–46, New York, NY, USA, 2008. ACM.

[5] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. The effectiveness of source code obfuscation: An experimental assessment. In *IEEE 17th International Conference on Program Comprehension (ICPC)*, pages 178–187, may 2009.

[6] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Dept. of Computer Science, The Univ. of Auckland, 1997.

[7] C. S. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation: tools for software protection. *IEEE Trans. Softw. Eng.*, 28:735–746, August 2002.

[8] K. Heffner and C. Collberg. The obfuscation executive. In *Information Security*, pages 428–440. Springer, 2004.

[9] M. H. Jakubowski, C. W. Saw, and R. Venkatesan. Iterated transformations and quantitative metrics for software protection. In *SECRYPT*, pages 359–368, 2009.

[10] M. Karnick, J. MacBride, S. McGinnis, Y. Tang, and R. Ramachandran. A qualitative analysis of java obfuscation. In *Proceedings of 10th IASTED International Conference on Software Engineering and Applications, Dallas TX, USA*, 2006.

[11] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 290–299, New York, NY, USA, 2003. ACM.

[12] I. Sutherland, G. E. Kalb, A. Blyth, and G. Mulley. An empirical examination of the reverse engineering process for binary files. *Computers & Security*, 25(3):221–228, 2006.

[13] H. Tamada, K. Fukuda, and T. Yoshioka. Program incomprehensibility evaluation for obfuscation methods with queue-based mental simulation model. In *Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pages 498–503, Aug 2012.