Towards Experimental Evaluation of Code Obfuscation Techniques

Mariano Ceccato[1], Massimiliano Di Penta[4], Jasvir Nagra[5], Paolo Falcarin[3],
Filippo Ricca[2], Marco Torchiano[3], Paolo Tonella[1]
[1]Fondazione Bruno Kessler–IRST, Trento, Italy
[2]Unità CINI at DISI, Genova, Italy
[3]Politecnico di Torino, Italy
[4]RCOST - Dept. of Engineering - University of Sannio, Benevento, Italy
[5]University of Trento, Italy
ceccato|tonella@fbk.eu, dipenta@unisannio.it, jas@nagras.com
paolo.falcarin|marco.torchiano@polito.it, filippo.ricca@disi.unige.it

ABSTRACT

While many obfuscation schemes proposed, none of them satisfy any strong definition of obfuscation. Furthermore secure general-purpose obfuscation algorithms have been proven to be impossible. Nevertheless, obfuscation schemes which in practice slow down malicious reverse-engineering by obstructing code comprehension for even short periods of time are considered a useful protection against malicious reverse engineering. In previous works, the difficulty of reverse engineering has been mainly estimated by means of code metrics, by the computational complexity of static analysis or by comparing the output of de-obfuscating tools. In this paper we take a different approach and assess the difficulty attackers have in understanding and modifying obfuscated code through controlled experiments involving human subjects.

Categories and Subject Descriptors

D.2.8 [**Metrics**]

General Terms

Security, Experimentation, Measurement

Keywords

Empirical studies, Software Obfuscation

1. INTRODUCTION

Source code obfuscation is widely used to prevent/limit malicious attacks to software systems conducted by decompiling and understanding or modifying source code. In particular client applications in distributed application represent a privileged target for this kind of attacks. For instance many networked Java applications are often built as two components, a server run in a controlled environment and a client, often distributed as bytecode and run in a virtual machine which can easily be decompiled, understood, and patched to behave according to attacker intentions. Despite the proved theoretical impossibility of building general purpose obfuscators [2], implementations of obfuscators exist and are used in practice. Available obfuscators provide limited though effective protection against malicious reverse engineering by making code hard to understand and/or difficult to analyze through automatic code analysis tools [9]. However, few works cope with the problem of measuring the extent of protection offered by state of the art code obfuscation techniques.

The notion of reverse engineering complexity of obfuscated code is captured by the intuitive notions of *potency* and *resilience*. Potency is the amount of obscurity added to the code, i.e. how much more complex to understand and to analyze is the obfuscated code with respect to the original one. Resilience measures, instead, how difficult it is to automatically break the obfuscation.

This paper details the definition, design and planning of a series of controlled experiments aimed at empirically assessing the capability of source code obfuscation techniques — namely identifier renaming and opaque predicates — to reduce the capability of a subject to successfully complete an attack, and to increase the effort needed for the attack. The study was performed asking subject to perform understanding tasks or change tasks on the decompiled (either obfuscated or clear) client code of client-server Java applications. We report preliminary results obtained in a first experiment performed with 8 graduate students and related to the effectiveness of the identifier renaming obfuscation on Java code.

In the past, evaluation of obfuscation has been mainly addressed through code metrics or by applying automatic de-obfuscators. In [15], the amount of time required to perform automatic de-obfuscation is used to evaluate the *control-flow flattening* obfuscation, relying on a combination of static and dynamic analysis. An attempt to quantify and compare the level of protection of several obfuscation techniques is presented by Anckaert et al. [1]. Their contribution is a series of metrics based on *code*, *control flow*, *data* and *data flow*. They computed such metrics on some case study applications (both on clear and obfuscated code) but no attempt has been made in the direction of validating such metrics. Our contribution is, instead, devoted to assess obfuscation empirically. We compare the performance of human subjects while performing attack tasks both on obfuscated and clear code.

The work more similar to ours is a study on the complexity of reverse engineering binary code, that involved human subjects [13]. A group of 10 students (of heterogeneous level of experience) has

QoP'08, October 27, 2008, Alexandria, Virginia, USA.

been asked to perform static analysis, dynamic analysis and modification tasks on several C compiled programs. While the students ability to successfully achieve the reverse engineering tasks showed a correlation with the student experience, no correlation was observed between ability and source code complexity metrics (Halstead and McCabe metrics). Our empirical evaluation has a different aim. We intend to measure the effect size of obfuscation, i.e., quantify the increased effort necessary to reverse engineer an obfuscated program, compared to the effort necessary for a program in clear. Such effect size is the fundamental metrics that may justify the adoption of available obfuscators. Moreover, it allows to compare alternative obfuscators in terms of increased tamper resistance.

The paper is organized as follows: Section 2 provides a primer on the obfuscation techniques we considered. Section 3 gives the details of the experimental design we used and Section 4 reports preliminary experimental results and threats to validity. Section 5 concludes the paper.

2. A PRIMER ON OBFUSCATION TECHNIQUES

This section briefly describes the code obfuscation techniques used in our experiments.

Obfuscation transformations are classified into three classes [4]: *layout obfuscations*, removing relevant information (such as identifier names) from the code without changing its behavior; *data obfuscations*, transforming application data and data structures (e.g., data encoding, data splitting); and, *control-flow obfuscations*, altering the original flow of the application. The most relevant techniques for the present work are *identifier renaming* and *opaque predicates*.

Identifier renaming is an instance of layout obfuscation that removes relevant information from the code by changing the names of classes, fields and operations into meaningless identifiers, so as to make it harder for an attacker to guess the functionalities implemented by different parts of the application. There are several features of identifier renaming which are worth noting. It is a widely implemented obfuscation technique, implemented by many commercial and academic obfuscators. The original identifiers are lost during renaming and in this sense the obfuscation is irreversible. With intelligent and human assisted analysis, one may be able to provide some meaningful identifiers, however, the original identifiers are lost. Identifier renaming also has no performance overheard.

Nevertheless much of the structure of the program is preserved which may assist an attacker during reverse-engineering. An extension of this technique was proposed by Tyma [14] where instead of renaming an identifier to a new meaningless one, identifiers are reused whenever possible but in such a way that overloading resolves the introduced ambiguity correctly.

Obfuscation based on *Opaque predicates* [5] is a control-flow obfuscation that tries to hide the original behavior of an application by complicating the control flow with artificial branches. An opaque predicate is a conditional expression whose value is known by the obfuscator, but is hard to deduce statically by an attacker. An opaquely True (False) predicate always evaluates to True (False) at a given position in a program. An opaque predicated can be used in the condition of a newly generated *if* statement. One branch of the *if* statement is filled with the original application code, while the other is filled by a bogus version of it. Only the former branch will be executed, causing the semantics of the application to remain the same. In order to generate resilient opaque predicates, pointer aliasing can be used, since inter-procedural static alias analysis is known to be intractable.

Table 1: Overview of the experiment.

Goal	To analyze the effect of source code obfuscation techniques
Quality focus	Capability of understanding the obfuscated code Capability to perform attacks on the obfuscated code
Context	Objects: two Java client-server applications: Car Race and Chat Subjects: Graduate students
Null hypotheses	H_{01}: no effect of obfuscation on understanding level H_{02}: no effect of obfuscation on the capability of performing a change task H_{03}: no effect on the time needed for comprehension H_{04}: no effect on the time needed for completing a change task
Main factor **Treatments**	Obfuscation Decompiled, obfuscated code vs. decompiled, clear code
Other factors	Subjects' Ability, System, Lab
Dependent variables	(i) Ability to perform comprehension tasks (ii) Time required for comprehension (iii) Ability to correctly perform a change task (iv) Time required to perform a change task

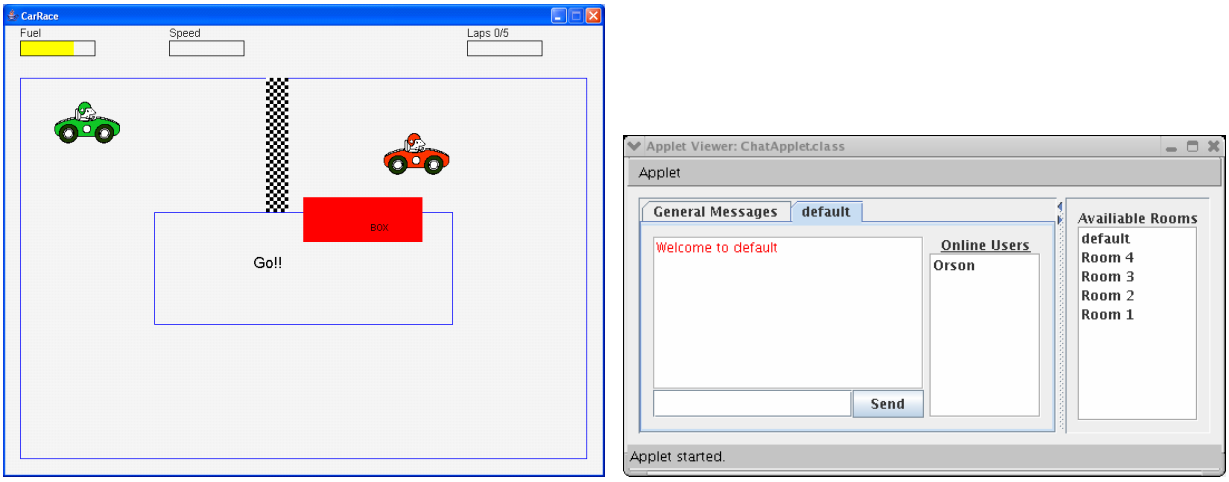
3. EXPERIMENT PLANNING

This section describes the definition, design and settings of the proposed experimentation following the Goal Question Metric template [3] and guidelines by Wohlin *et al.* [16] and [11]. The study definition is summarized in Table 1.

The *goal* of this empirical study is to analyze the effect of source code obfuscation techniques with the *purpose* of evaluating their effectiveness in making the code resilient to malicious attacks. The *quality focus* regards how the obfuscation reduces the attacker capability to understand and modify the source code, and above all how the obfuscation increases the effort needed to successfully complete an attack. This is a crucial point in our experimentation: although we are aware that an attacker could be able to complete an attack on obfuscated code anyway, she/he could be discouraged if such an attack requires a substantial effort/time. Results of this study can be interpreted from multiple *perspectives*:

1. a researcher would be interested to assess an obfuscation technique. Most existing assessments (e.g., [1, 8]) are based on metrics, estimating the increased code complexity, or on arguments about the increased difficulty of static code analysis (e.g., computational complexity of static alias resolution involved in a successful attack). Only a few works are based on attacks performed by human subjects [13] and none, to the authors' knowledge, applied rigorous approaches, such as those available from the area of empirical software engineering [11].

2. a practitioner, who wants to ensure high resilience to attacks to the part of a distributed application delivered to the clients, running in an untrusted environment.



(a) Car Race (b) Chat

Figure 1: Screen-shots of the two systems used in the experiment.

The *context* of this study consists of *subjects* involved in the experimentation and playing the role of attackers, and *objects*, i.e., systems to be attacked. Subjects are mainly graduate students, i.e., either Master or PhD students. In the first experiment we performed subjects are 8 Master students from the computer science degree of University of Trento. Subjects have a good knowledge on Java programming (they previously developed non-trivial systems as projects for at least 3 exams), and an average knowledge about software engineering topics (e.g., design, testing, software evolution). The subjects attended at least one software engineering course where they learned analysis, design and testing principles.

The objects used to conduct the experiment are two client-server applications developed in Java, a *Car Race* game and a *Chat* system.

CarRace is a network game that allows two players to run a car race (Figure 1-a). The player that first completes the total number of laps wins the race. During the race players have to refuel at the box. The number of completed laps and the fuel level is displayed on the upper part of the window. The client consists of 14 classes, for a total of 1215 LOC.

ChatClient (Figure 1-b) is a network application that allows people to have text based conversation through the network. Conversations can be public or private, depending on how they are initiated. The application shows on the right a list of available rooms. When the application starts, the "default" room is accessed. It is a public room where all the users are participating. In order to access another room, the name of the room must be clicked from the "Available Rooms" list, a new tab will be visualized. All the messages sent to a conversation within a room are received by all the users registered on that room. A private conversation (involving only two users) can be initiated by clicking the name of a user from the "Online Users" list. The client consists of 13 classes, for a total of 1030 LOC.

3.1 Hypotheses formulation and variable selection

Following the study definition above reported, we can formulate four research questions that will be addressed in the study:

RQ1: To what extent the obfuscation reduces the capability of subjects to comprehend decompiled source code?

RQ2: To what extent the obfuscation increases the time needed to perform a comprehension task?

RQ3: To what extent the obfuscation reduces the capability of subjects to perform a change task?

RQ4: To what extent the obfuscation increases the time needed to perform a change task?

Once research questions are formulated, it is possible to turn them into null hypotheses that can be tested in an experiment:

- H_{01} The obfuscation does not significantly reduce source code comprehensibility.
- H_{02} The obfuscation does not significantly increase the time needed to perform code comprehension tasks.
- H_{03} The obfuscation does not significantly reduce the capability of subjects to correctly perform a change task.
- H_{04} The obfuscation does not significantly increase the time needed to perform a change task.

The four hypotheses are *one-tailed*, since we are interested in analyzing the effect of obfuscation in one direction, i.e., to investigate whether the obfuscation *reduces* the capability to understand the code and to perform a change task, and whether it *increases* the time needed for such tasks.

The above null hypotheses suggest we have four dependent variables, i.e. *comprehension level*, *time needed for comprehension*, *success of the change task*, and *time needed to perform the change task*. To measure the attacker's capability comprehension level, we asked subjects to run the application, look at the client source code, perform two comprehension tasks, reported in Table 2. For each task subjects had to provide an answer. Tasks were conceived so that only one correct answer is possible, thus correct answers were evaluated as one, wrong answers as zero. To measure the capability to perform an attack, we asked subjects to perform two change tasks, reported in Table 3 for the two different systems. Since attacks can be thought as maintenance tasks, we evaluated the correctness of the attack by running test cases on the changed code

Table 2: Comprehension tasks

CarRace	T1	In order to refuel the car has to enter the box. The box area is delimited by a red rectangle. What is the width of the box entrance (in pixel)?
	T2	When the car crosses the start line, the number of laps is increased. Identify the section of code that increases the number of laps the car has traveled (report the class name/s and line number/s with respect to the printed paper sheets).
Chat	T1	Messages going from the client to the server use an integer as header to distinguish the type of the message. What is the value of the header for an outgoing public message sent by the client?
	T2	When a new user join, the list of the displayed "Online users" is updated. Identify the section of code that updates the list of users when a new user joins (report the class name/s and line number/s with respect to the printed paper sheets).

Table 3: Change tasks

CarRace	T3	The car can run only on the track and obstacles have to be avoided, if a wall is encountered the car stops. Modify the application such that the car can take a shortcut through the central island.
	T4	The fuel constantly decreases. Modify the application such that the fuel never decreases.
Chat	T3	Messages are sent to a give room, if the user is registered in the room and if the message is typed in the corresponding tab. Modify the application such that all the messages from the user go to "Room 1" without the user entering the room.
	T4	Messages are sent and displayed with the timestamp that marks when they have been sent. Modify the application such that the user sends messages with timestamp equals to 3,00 PM.

subjects sent us back, and evaluated the change as successful if test cases passed. A test case was defined for each change task, using a black-box strategy. They consists in sending an appropriate sequence of graphic events to delivered client, and verifying whether the required changes have been performed correctly using a special testing server.

The main factor of the experiment—that acts as an independent variable—is the obfuscation (identifier renaming). Such a factor can have two treatment levels, i.e., subjects can perform the comprehension and change on obfuscated, decompiled source code or on clear, decompiled source code (control groups used for comparison purposes).

The results can be affected by other factors, such as:

- *the System* used in the comprehension/maintenance task: as detailed in Section 3.2, to use a balanced design we need two objects. Although they are comparable in complexity, subjects can perform differently on different systems;
- the *subjects' Ability*, we have assessed considering bachelor grade and previous exams grades. According to the assessment made, we have classified subjects in *High* and *Low* ability subjects.
- the *Lab*: since the experiment requires two laboratories, a learning effect is possible from labs, and it should be analyzed.

3.2 Experiment design

We adopted a balanced experiment design intended to fit two Lab sessions (2-hours each). Subjects were split into four groups, each one working in Lab 1 on a system with a treatment and working in Lab 2 on the other system with a different treatment (see Table 4).

Table 4: Experiment design(O) = Obfuscated, (C) = Clear.

	Group A	**Group B**	**Group C**	**Group D**
Lab 1	CarRace (O)	CarRace (C)	Chat (C)	Chat (O)
Lab 2	Chat (C)	Chat (O)	CarRace (O)	CarRace (C)

The design ensures that each subject worked on different *Systems* in the two *Labs*, receiving each time a different treatment. Also, the design permits us to consider different combinations of *System* and treatment in different order across *Labs*. More important, the chosen design permits the use of statistical tests (Two-Way and Three-Way ANOVA[6]) for studying the effect of multiple factors. Subjects were split into four groups making sure that *high* and *low Ability* subjects were equally distributed across groups.

3.3 Experiment material and procedure

This section details the procedure we followed to perform the study. Before the experiment, subjects were properly trained with lectures on obfuscation techniques, and with exercises having the purpose of performing comprehension tasks on the (non-obfuscated) source code of a simple vending machine system. Right before the experiment, we provided to subject a detailed explanation of the tasks to be performed during the lab, without however explicitly the study hypotheses, to avoid a possible bias.

To perform the experiment, subjects used a personal computer with the Eclipse development environment—which they are familiar with—, including notably syntax highlighting and debugger, and the Java API documentation available. We distributed to subjects the following material:

- a short textual documentation of the system (CarRace or Chat) they had to attack, comprising installation instructions;
- a jar archive containing the server of the application. As mentioned above, both CarRace and Chat are client-server applications; to avoid during the experimentation problems due to reduced bandwidth or limited Internet access (often due to University laboratory restrictions) we let the subjects running the server locally, without providing the source code and checking they do not decompile it;
- the decompiled client source code, either clear or obfuscated depending on the group the subject belonged to (Table 4);
- printouts of the slides explaining the experiment procedure, that we presented just before distributing the material.

The experiment was carried out according to the following procedure. Subjects had to:

1. read the application description;
2. import the client source code in Eclipse;
3. run the application to familiarize with it;
4. for each of the four tasks to be performed:
   (a) ask the teacher a paper sheet describing the task to be performed;
   (b) mark the start time;
   (c) read the task and perform it;
   (d) write the answer (for comprehension tasks);
   (e) mark the stop time and return the paper sheet;

5. after completing all tasks, create an archive containing the maintained project and send it to the teacher by email;

6. complete a survey questionnaire.

During the experiment, teaching assistants and professors were in the laboratory to prevent collaboration among subjects, and to check that subjects properly followed the experimental procedure, i.e., they performed the tasks in the given order, and they correctly annotated the time spent.

After the experiment, subjects had to fill a post-experiment survey questionnaire. It aimed at both gaining insights about the students' behavior during the experiment and finding justifications for the quantitative results. The questionnaire contains questions — most of them expressed in a Likert scale [10] — related to:

- the clarity of tasks and objectives;
- the difficulties experienced when performing the different tasks (comprehension and change);
- the confidence in using the development environment and the debugger;
- the percentage of total time spent on looking the source code, on executing the system (such questions foresee as possible answer;
- (for subjects having obfuscated code only) to what extent they considered the analysis of obfuscated code hard;
- (for subjects having obfuscated code only) whether subjects considered important executing the system to better understand its behavior.

3.4 Analysis method

Different kinds of statistical tests needs to be used to analyze results of this experiment. To analyze whether the obfuscation reduces the correctness of comprehension and change tasks, we need to use tests on categorical data (i.e., the tasks can be either correct or wrong). In particular, we used the Fisher's exact test [7], more accurate than χ^2 test for small sample sizes, which is another possible alternative to test the presence of differences in categorical data. Then, two non-parametric tests are used to test the hypotheses related to differences in time need to perform the tasks. Non–parametric statistics are used since they do not require any assumption on the underlying population distribution. First, an unpaired analysis — i.e., an analysis of all data grouped by different treatments of the main factor — is performed using the Mann-Whitney one-tailed test[6]. Given the chosen experiment design, it is also possible to use a paired test, i.e., the Wilcoxon test[6]. Such a test allows to check whether differences exhibited by subjects with different treatments (clear and obfuscated code) over the two labs are significant.

While the above tests allows for checking the presence of significant differences, they do not provide any information about the magnitude of such a difference. This is particularly relevant in our study, since we are interested to investigate to what extent the use of obfuscation reduces the likelihood of completing an attack, and what is the increment of the time needed for the attack. To this aim, two kinds of effect size measures are used, the *odds ratio* and the Cohen d effect size.

The odds ratio is a measure of effect size that can use for dichotomic categorical data. An odds [12] indicate how much likely is that an event will occur as opposed to it not occurring. Odds ratio is defined as the ratio of the odds of an event occurring in one group (e.g., experimental group) to the odds of it occurring in another group (e.g., control group), or to a sample-based estimate of that ratio. If the probabilities of the event in each of the groups are indicated as p (experimental group) and q (control group), then the odds ratio is defined as:

$$OR = \frac{p/(1-p)}{q/(1-q)}$$

An odds ratio of 1 indicates that the condition or event under study is equally likely in both groups. An odds ratio greater than 1 indicates that the condition or event is more likely in the first group. Finally, an odds ratio less than 1 indicates that the condition or event is less likely in the first group.

The Cohen d effect size indicates the magnitude of a main factor treatment effect on the dependent variables (the effect size is considered small for d$\geq$0.2, medium for d$\geq$0.5 and large for d$\geq$0.8). For independent samples, it is defined as the difference between the means (M_1 and M_2), divided by the pooled standard deviation:

$$d = (M_1 - M_2)/\sigma$$

The experiment results could be influenced by any of the confounding factors described in Section 3.1. To this aim, a two-way Analysis of Variance (ANOVA) can be used to analyze the effect of confounding factors –mainly System, Lab, and Ability– on the main factor (the *Method*) and their interaction. In general ANOVA is considered quite robust also for non normal and non interval-scale variables.

Table 5: Change tasks

	Comprehension		Change		Overall	
Treatment	Wrong	Correct	Wrong	Correct	Wrong	Correct
Clear	7	11	3	15	10	26
Obfuscated	12	8	12	8	24	16

4. PRELIMINARY RESULTS

4.1 Data analysis

This section reports preliminary results we obtained from a first experiment we performed with 8 subjects from the University of Trento, using *identifier renaming* as the obfuscation method. We only report preliminary analyses aimed at testing the hypotheses formulated in Section 3.1. Analysis of confounding factors and survey questionnaires as well as other analyses (e.g., paired analysis per subject) will be reported in future work. Table 5 reports — for the 8 subjects — the number of correct and wrong tasks related to comprehension, change and to the overall experiment. Differences between overall number of tasks (clear Vs obfuscated) is due to missing answers.

We tested the presence of a significant difference between the two different treatments using the Fisher test, and found no significant difference for comprehension (p-value=0.33), while a significant difference was found for the change (p-value=0.009). Also considering the whole set of tasks (including both comprehension and change) a significant difference (p-value=0.006) is visible.

More important than testing the presence of differences is computing the odds ratio, i.e., the ratios between *odds* of correct answers for subjects having obfuscated or clear code. For comprehension tasks the odds ratio is 2.3, meaning that subjects having obfuscated code have less than half odds of subject having clear code to successfully complete the tasks. The odds ratio is higher

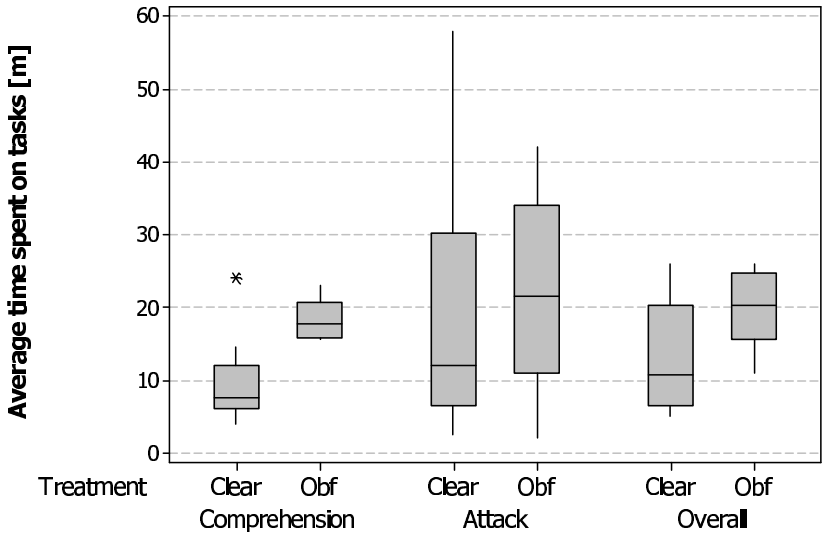


Figure 2: Boxplots of time needed to perform the tasks.

for change tasks (7.1) indicating how the presence of obfuscated code substantially reduce seven times the odds of completing the attack. Finally, the odds ratio for the overall data set is 3.8.

Figure 2 reports boxplots of time needed to complete the tasks. Time is computed, for each subject, as the average time over comprehension tasks, change tasks and overall tasks the subject was able to complete. For this preliminary analysis we do not consider yet whether the task was also correctly performed, while we intend to do this refined analysis in our future work. The Mann-Whitney, one-tailed test indicated a significant difference for comprehension time (p-value=0.002), while no significant difference was found for change tasks time (p-value=0.19). A significant difference is visible if considering all the tasks (p-value=0.02). The effect size is large for comprehension tasks (d=1.8), small for change tasks (d=0.2), and again large if considering both the task types (d=1.03).

Results suggest a rejection for H_{02} (time needed for comprehension), H_{03} (correctness of change tasks) while it is not possible to reject H_{01} (correctness of comprehension tasks) and H_{04} (time needed for change tasks). Thus, it seems that subjects having obfuscated code spent significantly more time when comprehending the code, however achieving results not far from those having clear code. On the other hand, while tampering with the code subjects having obfuscated code did not exhibit any time overhead, while they had a significantly lower probability of successfully completing the task.

4.2 Threats to validity

We identified the main threats to the validity [16] of our results: construct, internal, conclusion, and external validity threats.

Construct validity threats concern the relationship between theory and observation. They are mainly due to the method used to assess the outcomes of tasks. The measurements we conceived — comprehension questions with one possible answer and test cases to assess code correctness — are as objective as possible.

Internal validity threats concerns external factors that may affect an independent variable. We controlled for different systems, labs, and subjects' ability, although for reasons of space we did not show analysis for these cofactors in this paper. Moreover the design is a full factorial design with random assignments that balances individual factors and learning effects.

Conclusion validity concerns the relationship between the treatment and the outcome. The statistical analysis is performed mainly using non-parametric tests that do not assume data normality.

External validity concerns the generalization of the findings. The main threat in this area stems from the type of subjects: in the reported experiment they are all master students. Only further studies can confirm whether the results obtained can be generalized to professional developers.

5. CONCLUSIONS

This paper provided the definition and planning of a series of controlled experiments we are carrying out. We aim at empirically assessing the capability of source code obfuscation techniques to make decompiled code resilient to comprehension and attack activities. Preliminary results we obtained with 8 graduate students indicates indeed that obfuscation reduces the capability of subjects to understand and modify the source code. In the reported experiment the effect appears particularly relevant for tasks requiring code modification: the odds of successfully completing the task are 7 times lower for subjects working with obfuscated code. The time needed to perform the tasks also significantly increases in presence of obfuscation, and this is particularly true for comprehension tasks, where a significant difference and a high effect size is visible.

At the time of writing we are analyzing data from further experiments we performed, involving larger sets of subjects and different source code obfuscation techniques. Also, we are performing more

accurate statistical analyses, considering also confounding factors and taking into account feedbacks provided by subjects through survey questionnaires. Future work will report detailed analyses and discussions of these experiments.

6. ACKNOWLEDGMENTS

This work was supported by funds from the European Commission (contract N^o 021186-2, RE-TRUST project) and Italian Government (grant PRIN2006-2006098097, METAMORPHOS project).

7. 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: Proceedings of the 2007 ACM workshop on Quality of protection*, pages 15–20, New York, NY, USA, 2007. ACM.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. *Lecture Notes in Computer Science*, 2139:19–23, 2001.

[3] V. Basili, G. Caldiera, and D. H. Rombach. *The Goal Question Metric Paradigm, Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

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

[5] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 184–196, New York, NY, USA, 1998. ACM.

[6] P. Dalgaard. *Introductory Statistics with R*. Springer, 2002.

[7] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press; 7 edition, 2007.

[8] H. Goto, M. Mambo, K. Matsumura, and H. Shizuya. An approach to the objective and quantitative evaluation of tamper-resistant software. In *Third International Workshop on Information Security (ISW2000)*, pages 82–96. Springer, 2000.

[9] K. Heffner and C. Collberg. The obfuscation executive. In *Proceedings of the 7th International Conference on Information Security, ISC'04*, volume 3255 of *LNCS*, pages 428–440, 2004.

[10] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London, 1992.

[11] S. L. Pfleeger. Experimental design and analysis in software engineering. *SIGSOFT NOTES, Parts 1 to 5*, 1994 and 1995.

[12] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[13] 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.

[14] P. Tyma. Method for renaming identifiers of a computer program. US patent 6,102,966, 2000.

[15] S. Udupa, S. Debray, and M. Madou. Deobfuscation: reverse engineering obfuscated code. *Reverse Engineering, 12th Working Conference on*, Nov. 2005.

[16] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.