Empirical Software Engineering (EMSE) manuscript No.
(will be inserted by the editor)

A Family of Experiments to Assess the Effectiveness and Efficiency of Source Code Obfuscation Techniques

**Mariano Ceccato, Massimiliano Di Penta,
Paolo Falcarin, Filippo Ricca,
Marco Torchiano, Paolo Tonella**

the date of receipt and acceptance should be inserted later

Abstract
Context: code obfuscation is intended to obstruct code understanding and, eventually, to delay malicious code changes and ultimately render it uneconomical. Although code understanding cannot be completely impeded, code obfuscation makes it more laborious and troublesome, so as to discourage or retard code tampering. Despite the extensive adoption of obfuscation, its assessment has been addressed indirectly either by using internal metrics or taking the point of view of code analysis, e.g., considering the associated computational complexity. To the best of our knowledge, there is no publicly available user study that measures the cost of understanding obfuscated code from the point of view of a human attacker.
Aim: this paper experimentally assesses the impact of code obfuscation on the capability of human subjects to understand and change source code. In particular, it considers code protected with two well-known code obfuscation techniques, i.e., identifier renaming and opaque predicates.
Method: We have conducted a family of five controlled experiments, involving undergraduate and graduate students from four Universities. During the experiments, subjects had to perform comprehension or attack tasks on decompiled clients of two Java network-based ap-

Mariano Ceccato, Paolo Tonella
Fondazione Bruno Kessler–IRST, Trento, Italy
E-mail: ceccato|tonella@fbk.eu

Massimiliano Di Penta
Department of Engineering – University of Sannio, Benevento, Italy
E-mail: dipenta@unisannio.it

Paolo Falcarin
School of Architecture, Computing and Engineering – University of East London, UK E-mail: falcarin@uel.ac.uk

Filippo Ricca
DIBRIS, University of Genova, Italy E-mail: filippo.ricca@disi.unige.it

Marco Torchiano
Politecnico di Torino, Italy E-mail: marco.torchiano@polito.it

plications, either obfuscated using one of the two technique, or not. To assess and compare the obfuscation techniques, we measured the correctness and the efficiency of the performed task.

Results: —at least for the tasks we considered—simpler techniques (i.e., identifier renaming) prove to be more effective than more complex ones (i.e., opaque predicates) in impeding subjects to complete attack tasks.

1 Introduction

Encryption and firewalls are classic solutions to mitigate the threat of remote attackers (i.e., Man-In-The-Middle) who try to break into software systems. However, these classic approaches do not help defending software systems when the attacker is the end user (i.e., Man-At-The-End [14]). A vast class of client applications are required to run under strict usage conditions, that may be violated on tampered clients. For example, on-line game providers should prevent cheating to ensure a fair competition; client applications for media conditioned-access (e.g., pay-per-view digital TV) could be tampered with to access the service in a way that was not intended by the service provider (e.g., paying a reduced fee). Other relevant examples of clients vulnerable to code tampering are rich Web 2.0 (Ajax) applications and apps for smart-phones and tablets.

Among the various techniques available for protecting code from different Man-At-The-End attacks, code obfuscation is one of the most popular choice, deployed to prevent code comprehension, the precondition for further code tampering. Obfuscation consists of code transformations that make a program more difficult to understand by changing its structure, while preserving the original functionalities. However, a determined attacker, after spending enough time to inspect obfuscated code, might locate the functionality to alter and succeed in her/his malicious purpose. For this reason, obfuscation is rarely deployed alone. Often, obfuscation is complemented by other approaches, such as code replacement/update [5] or tamper-detection with self-checkers [6] [19] or protections update [28] [15], in order to give an attacker a limited amount of time to complete her/his intent. However, to properly plan code updates, the provider should estimate how long obfuscation would resist, i.e., the time an attacker needs to understand obfuscated code.

Despite code obfuscation is a largely adopted solution [9], and many different obfuscation approaches have been proposed [8], there are no publicly available user studies on code obfuscation that compare different obfuscation techniques and measure how long it takes for an attacker to understand and change obfuscated code.

In this paper we present a family of controlled experiments, planned and conducted using a rigorous approach as described by Wohlin *et al.* [33], to measure the level of protection offered by code obfuscation. Five experiments have been designed and conducted involving overall 74 students with different levels of experience (e.g., undergraduate, master, and PhD students). Subjects were asked to perform understanding and change tasks on code protected with two of the most prominent approaches for code obfuscation—identifier renaming and opaque predicates—and their performance has been assessed and compared in terms of task correctness and efficiency.

The work presented here extends the one presented in [3] by providing the following new contributions: (1) a new treatment, namely Opaque predicates obfuscation; (2) results from three further experiments (Exp III, Exp IV and Exp V); (3) an extended data analysis; (4) an extended discussion on achieved results.

Results shed light on the validity and on the limits of code obfuscation, clarify which strategies attackers adopt to try and break it, explained whether tools are helpful, and made clear whether the attacker's experience plays any role. Experimental outcomes allow us to quantify the expected delay of successful attacks, depending on which obfuscation is employed.

The paper is organized as follows: Section 2 covers the background on code obfuscation. Then, Section 3 presents the experimental design and planning. Experimental results are reported in Section 4 and then discussed in Section 5. Eventually, related works and conclusions close the paper, respectively in Section 6 and Section 7.

2 A Primer on Source Code Obfuscation Techniques

Obfuscation transformations can be classified into three main classes [9]: *layout obfuscations*, *control-flow obfuscations* and *data obfuscations*.

Layout obfuscations remove relevant information (such as identifier names) from the code without changing its behavior. *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, offered by several 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 reintroduce some meaningful identifiers. However, the original identifiers are lost. Identifier renaming has no performance overheard. An extension of the basic identifier renaming technique was proposed by Tyma [31], 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. The main weakness of this obfuscation technique is that much of the structure of the program is preserved, which may assist an attacker during reverse-engineering.

We applied Identifier renaming obfuscation on the bytecode using the *SandMark* tool[1], which replaces identifiers with randomly generated ones. Obfuscated bytecode is decompiled into Java source code using the Jad 1.5 decompiler[2].

Control-flow obfuscations alter the original flow of the application. Obfuscation based on *opaque predicates* [10] 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 predicate 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 with 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 [16].

Available tools for opaque predicates apply directly on byte-code, but the obfuscated byte-code makes decompilers fail. So, we applied opaque predicates by means of a source-to-source transformation program implemented in TXL [11], similarly to what described

[1] http://sandmark.cs.arizona.edu/

[2] http://www.kpdus.com/jad.html

by Collberg *et al.* [10]. For this reason the results of our study are related (and limited to) *decompilable* opaque predicates. To this aim, a pointer-intensive dynamic data structure is created and a set of pointers on this structure are maintained. Opaque predicates are alias conditions between pointers on such data structure. They are evaluated at decision points, in order to hide the correct execution flow to the attacker. Moreover, random and buggy code is added into the basic blocks controlled by those branches that are never taken due to opaque predicates.

As part of the obfuscation, new statements are added to the code to continuously update the data structure. Update statements, while mutating the data structure, guarantee a known subset of alias conditions to remain valid. Nodes are added, removed and updated, so that aliases among pointers are frequently changed, thus making it very hard to statically detect whether two pointers refer to the same entity, even with the support of automatic analysis tools.

Data obfuscations transform application data and data structures (e.g., data encoding, data splitting).

In this paper we studied two out of these three obfuscations, i.e. identifier renaming and opaque predicates.

3 Experimentation Definition and Planning

This section reports the definition, design and settings of the experiments in a structured way, following the template and guidelines by Wohlin *et al.* [33].

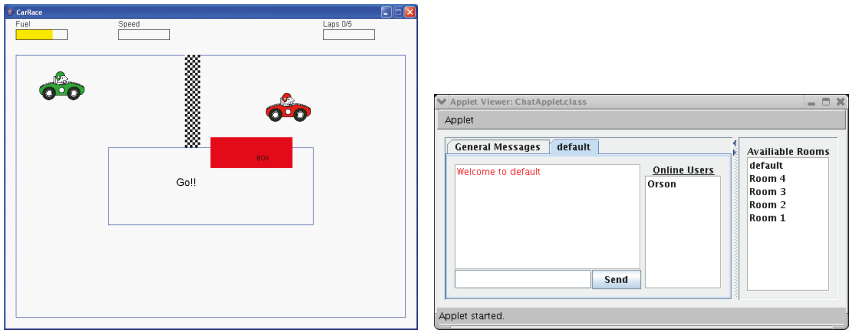
The *goal* of this study is to analyze the effect of two source code obfuscation techniques, named *identifier renaming* and *opaque predicates*[3] with the *purpose* of evaluating their ability in making the code resilient to malicious attacks. The *quality focus* regards how these obfuscation techniques reduce the attacker's capability to correctly and efficiently understand and modify the source code. Investigating the effect of obfuscation on the attack efficiency 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 could be discouraged if such an attack requires a substantial effort/time. Results of this study can be interpreted from multiple *perspectives*: (i) a researcher interested to empirically assess the identifier renaming and opaque predicates obfuscation techniques; and (ii) a practitioner, who wants to ensure high resilience to attacks to some components of a distributed application delivered to the clients, running in an untrusted environment.

3.1 Context: the Subjects

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 University students, either Bachelor, Master or PhD students. The study consists of five experiments, involving in total 74 students:

- Exp I was performed with 10 Master students from University of Trento;
- Exp II with 22 PhD students from Politecnico di Torino;
- Exp III with 16 Master students from University of Sannio;
- Exp IV with 13 (different) Master students from University of Trento; and

[3] As already mentioned in Section 2, we restrict to *decompilable* opaque predicates.



(a) CarRace (b) ChatClient

Fig. 1 Screenshots of the object applications.

– Exp V with 13 Bachelor students from University of East London.

Bachelor students have just basic notions of programming in Java and some initial knowledge of software engineering (e.g., design, testing). Master students from both University of Trento and Sannio have an average knowledge about software engineering topics a good knowledge on Java programming. In fact, they previously developed non-trivial systems as projects for at least three exams, such as web applications and data processing programs, consisting in most of the cases of few thousands lines of Java code. The purpose of such projects was to pass the corresponding course.

All subjects attended at least one software engineering course where they learned analysis, design and testing principles. Most PhD students held a Master in Computer Engineering; a few were carrying out research in the field of Electronic Engineering.

3.2 Context: the Objects

The systems used to conduct the experiment are two client-server applications developed in Java, a *CarRace*[4] game and a *ChatClient*[5] system.

CarRace is a network game that allows two players to run a car race; a screenshot is shown in Fig. 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 1,215 LOC. When obfuscated with identifier renaming, system size does not change, while when using opaque predicates the obfuscated application grows to 3,783 LOC.

ChatClient is a network application that allows people to have text based conversations through the network; a screenshot is shown in Fig. 1 b). Conversations can be public or private. The client consists of 13 classes, for a total of 1,030 LOC of clear code or code obfuscated with identifier renaming. When using opaque predicates the client reaches 3,642 LOC.

These systems are comparable in complexity and size. In addition, they are small enough to fit the time constraints of our experiments, and they are also realistic for small/medium sized comprehension tasks.

[4] CarRace was developed by one of the authors as case study application for a previous work [5]

[5] ChatClient is an open source project available at http://sourceforge.net/projects/jchat

Table 1 Comprehension and change 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 completed (report the class name/s and line number/s with respect to the printed paper sheets).
	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.
ChatClient	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 joins, 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).
	T3	Messages are sent to a given 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 a constant timestamp = 3,00 PM.

3.3 Hypothesis Formulation and Variables Selection

Following the study definition reported above, we can formulate the following null hypotheses to be tested:

- on *Identifier renaming*:
  - H_{01} *identifier renaming* obfuscation does not significantly decrease the capability of an attacker to perform a *comprehension* task.
  - H_{02} *identifier renaming* obfuscation does not significantly decrease the capability of an attacker to perform a *change* task.
- on *opaque predicates*:
  - H_{03} *opaque predicates* obfuscation does not significantly decrease the capability of an attacker to perform a *comprehension* task.
  - H_{04} *opaque predicates* obfuscation does not significantly decrease the capability of an attacker to perform a *change* task.
- and on their comparison:
  - H_{05} there is no difference between *identifier renaming* and *opaque predicates* in decreasing the capability of an attacker to perform a *comprehension* task.
  - H_{06} there is no difference between *identifier renaming* and *opaque predicates* in decreasing the capability of an attacker to perform a *change* task.

Hypotheses H_{01}, H_{02}, H_{03} and H_{04} 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 attacker's capability to understand the source code and to perform a change task. Instead, hypotheses H_{05} and H_{06} are *two-tailed*, because in principle we do not know which obfuscation makes the code more difficult to understand and change.

The null hypotheses suggest we have two dependent variables, i.e., the *capability of performing comprehension tasks*, and the *capability of performing change tasks*. To measure the subject's capability to perform a comprehension task (*achieved comprehension level*), we asked subjects to run the application, look at the client source code, and perform two comprehension tasks, (T1 and T2 in Table 1). These tasks are conceived so that only one correct answer is possible, thus correct answers can be evaluated as one, wrong answers as zero. To measure the success subjects had in change tasks (*success of change tasks*), we

asked them to execute two change tasks (T3 and T4 in Table 1) against the two different systems. It is important to note that the proposed tasks are representative of realistic attacks that a hacker could perform on a distributed game or on a chat e.g., to gain unlimited fuel, or to get access to restricted messages. Since attacks can be thought of as maintenance tasks, we evaluated the correctness of the attack by running test cases on the code modified by the subjects, and evaluated the attack as successful if test cases passed (a similar approach has been used in a previous empirical study [26]). A test suite was defined for each change task. The test suite reproduces the interaction scenario of the attack to be performed and fails if the tampered behaviour is not observed.

Hypotheses are formulated abstractly in terms of the capability of attackers to complete attacks. However, to practically measure the capability of attackers, we have to resort to more concrete and measurable concepts; they are correctness, response time and efficiency. The capability of a subject in performing comprehension or change tasks is evaluated using three metrics:

- **Correctness** of comprehension/change tasks. The correctness $Corr_i = 1$ if the *i-th* comprehension or change task was correctly performed, 0 otherwise. Such a correctness assessment was performed by one of the authors who inspected the provided answers (comprehension tasks) and by running test cases (change tasks).
- **Time** to correctly perform comprehension/change tasks. We collected such information by asking subjects to fill in—while performing the experiment tasks—start and end time of each task. Such a variable is particularly important in this experiment because, although obfuscation might not totally prevent an attack, at least it could make it slower, thus discouraging the attackers or allowing system administrators to enact countermeasures. Precisely, the variable $Time_i$ accounts for the time (measured in minutes) needed to perform the task i. However, we perform statistics only $\forall i : Corr_i = 1$.
- **Efficiency** in performing comprehension/change tasks. It measures the number of correctly performed task per minute, and it is defined as:

$$\frac{\sum_{i=1}^{2} Corr_i}{\sum_{i=1}^{2} Time_i} \tag{1}$$

  As it can be noticed to the above formula, the efficiency sums over all (two) comprehension or change tasks.

The main factor of the experiment—that acts as our independent variable—is the presence of the treatment during the execution of the task. Different pairs of alternative treatments are used in different experiments, and they are summarized in Table 2, together with the hypotheses tests in each experiment. In Exp I and Exp II the two alternative treatments are (i) decompiled source code[6], derived from code obfuscated with identifier renaming, and (ii) decompiled clear code. In Exp III the two treatment are (i) decompiled source code, derived from code obfuscated with opaque predicates, and (ii) decompiled clear code. In Exp IV and Exp V the two alternative treatments are decompiled source code, derived from code obfuscated (i) with identifier renaming, and (ii) with opaque predicates.

Among the co-factors that can potentially affect the results, we identified and measured the following ones:

- *The subjects' Experience*, i.e., Bachelor, Master or PhD students. It is important to note that, although in this paper we analyze the effect of such a co-factor, and although we

[6] Subjects used decompiled code rather than source code because, in a realistic attack, they cannot access the source code, but they can decompile the binary or the bytecode

Table 2 Summary of the experiments. IR = Identifier renaming, OP = Opaque predicates.

Experiment	Hypotheses	Treatment 1	Treatment 2	Experience	University	# of Subjects
Exp I	H_1 H_2	Clear	IR	Master	Trento	10
Exp II	H_1 H_2	Clear	IR	PhD	Torino	22
Exp III	H_3 H_4	Clear	OP	Master	Sannio	16
Exp IV	H_5 H_6	IR	OP	Master	Trento	13
Exp V	H_5 H_6	IR	OP	Bachelor	London	13

Table 3 Experiment design. (TR_1) = Treatment 1, (TR_2) = Treatment 2.

	Group A	**Group B**	**Group C**	**Group D**
Lab 1	CarRace (TR_1)	CarRace (TR_2)	ChatClient (TR_2)	ChatClient (TR_1)
Lab 2	ChatClient (TR_2)	ChatClient (TR_1)	CarRace (TR_1)	CarRace (TR_2)

performed different experiments involving subjects with different experience, we cannot control it. This is because we performed a convenience sampling (based on subjects that volunteered to perform the experiment). Therefore, randomization on this co-factor is not possible.

- *The System* to be attacked: as detailed in Section 3.2, to use a balanced design we considered two systems: ChatClient and CarRace. Although their clients are comparable in terms of size and complexity, subjects may perform differently on different systems.
- *The Lab*, i.e., whether there is a learning effect across subsequent experiment laboratories.
- *Learning across subsequent tasks*: in the same way as for the Lab, we analyze whether there is a learning effect as subjects perform the four subsequent tasks.

While working on software projects, subjects undertake maintenance tasks that require to understand and change code written by other developers. The more experience subjects have in maintenance, the easier for them is to solve similar tasks. Maintenance activities are similar to the tasks addressed during experimental sessions. Therefore, experience is a co-factor that could influence the capability of subjects to successfully complete comprehension and change tasks. As subjects involved in the study hold a different level of experience, it makes sense to investigate its impact on tasks. The measure is reliable, because authors of this work were the professors in charge of the courses that hosted the experiments.

For each co-factor, we test (see Section 3.6) the effect on the efficiency in performing the attack—as defined in equation (1)—and the interaction with the main factor's treatments. In other words, we want to assess if co-factors influence the efficiency of subjects in performing an attack, and if they interact with the treatment to influence efficiency.

3.4 Experiments Design

We adopt a counter-balanced experiment design [21,33] intended to fit two Lab sessions (2-hours each). Subjects are classified 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 3). However, subjects work on their own, without any collaboration within the group. The design ensures that each subject works on different *Systems* in the two *Labs*, receiving each time a different treatment. Also, the design allows for considering different

combinations of *System* and *Treatment* in different order across *Labs*. More important, the chosen design permits the use of statistical tests (e.g., the permutation test, a non-parametric alternative to ANOVA) for studying the effect of multiple factors [20].

3.5 Experimental procedure

This section details the procedure we followed to perform the experiments, and the material employed. Before each experiment, subjects were properly trained with lectures on obfuscation techniques and with program comprehension exercises on the (non-obfuscated) source code of an electronic record book. The purpose of training is to make subjects confident about the kind of tasks they are going to perform and the environment (e.g., IDE and documentation) they will have available.

Right before the experiment, we provided subjects with a detailed explanation of the tasks to be performed during the lab; no reference was made to the study hypotheses.

For the experiment, subjects used a personal computer equipped with the Eclipse™ development environment—which they are familiar with—including syntax highlighting and debugger, and with the Java API documentation available. We distributed the following material, available online for replication purposes[7] to our subjects:

- a short textual documentation of the system they had to attack;
- a jar archive containing the server of the application. The server was executed locally by the subjects to avoid any network related problem. However, we did not provide the source code and checked that subjects did not decompile it;
- the decompiled client source code, either clear or obfuscated depending on the group the subject belonged to (see Table 2 and Table 3); and
- slides explaining the experiment procedure.

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 (CarRace or ChatClient) to familiarize with it;
4. for each of the four tasks to be performed: (i) ask the teacher for a paper sheet describing the task to be performed; (ii) mark the start time; (iii) read the task and perform it; and (iv) mark the stop time and return the paper sheet;
5. after completing all tasks, create an archive containing the modified project and send it to the teacher by email;
6. complete a post-experiment 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.

After the experiment, subjects were required to fill a post-experiment survey questionnaire. It aimed at both gaining insights about the subjects' behavior during the experiment and finding justifications for the quantitative results. The questionnaire contains 18 questions (see Table 4 and experimental package or a longer technical report [4] for details)—most of them expressed in a Likert scale [23] with 5 levels—related to:

- the clarity of tasks and objectives (Q1 – Q4);

[7] http://selab.fbk.eu/ceccato/replication_packages/id_renaming_vs_opaque_predicates_package.tgz

Table 4 Post-experiment survey questionnaire.

ID	Question
Q1	I had enough time to perform the tasks. (1–5).
Q2	The system description was clear. (1–5).
Q3	The lab objectives were clear. (1–5).
Q4	The tasks were perfectly clear. (1–5).
Q5	I experienced no difficulty in program understanding. (1–5).
Q6	I experienced no difficulty in the identification of the segment of code relevant for the tasks. (1–5).
Q7	I experienced no difficulty in changing the segment of code relevant for the tasks. (1–5)
Q8	I experienced no difficulty in using the development environment (Eclipse). (1–5)
Q9	I used the Eclipse debugger (never, only a few times, sometimes, often, always)
Q10	I experienced no difficulty in using the Eclipse debugger. (1–5)
Q11	The debugging environment is useful to execute the tasks. (1–5)
Q12	I found the renaming facility useful. (1–5)
Q13	How many executions (i.e., run of the System not in debugging mode) have you done on average before having completed the requirement? (1, $\geq$ 2 and $<$ 4, $\geq$ 5 and $<$ 7, $\geq$ 7 and $<$ 10, $\geq$ 10)
Q14	How many executions (i.e., run of the System in debugging mode) have you done on average before having completed the requirement? (1, $\geq$ 2 and $<$ 4, $\geq$ 5 and $<$ 7, $\geq$ 7 and $<$ 10, $\geq$ 10)
Q15	How much time (in terms of percentage) did you spend looking at the code? ($<$20%, $\geq$20% and $<$40%, $\geq$40% and $<$60%, $\geq$60% and $<$80%, $\geq$80%)
Q16	How much time (in terms of percentage) did you spend running the system? ($<$20%, $\geq$20% and $<$40%, $\geq$40% and $<$60%, $\geq$60% and $<$80%, $\geq$80%)
Extra questions for groups working on obfuscated source code:	
Q17	Understanding the obfuscated code is hard. (1–5)
Q18	Running the system is useful to understand the obfuscated code. (1–5)

1 = strongly agree, 2 = agree, 3 = not certain, 4 = disagree, 5 = strongly disagree.

- the difficulties experienced when performing the different tasks (comprehension, feature location[8], and change tasks) (Q5 – Q7);
- the confidence in using the development environment and the debugger (Q8, Q10);
- the usefulness of the Eclipse renaming and debugging features (Q11, Q12);
- debugger frequency of use (Q9), number of executions in debugging mode (Q14) and execution mode (Q13);
- the percentage of total time spent looking at the source code and executing the system (Q15, Q16);
- to what extent subjects considered the analysis of obfuscated code hard (Q17);
- whether subjects considered executing the system important to better understand the behavior of obfuscated code (Q18).

3.6 Analysis Method

Different kinds of statistical tests have been used to analyze the results of this experiment. All of them have been applied using the *R* statistical environment [24].

To analyze whether the obfuscation reduces the *correctness* of comprehension and change tasks, we used tests on categorical data (i.e., the tasks can be either correct or wrong). In particular, we used the Fisher's exact test [12], more accurate than the χ^2 test for small sample

[8] The goal of feature location [13] is to identify the computational units (e.g., procedures, class methods) that specifically implement a feature (e.g., requirement) of interest.

sizes, which is another possible alternative to test the presence of differences in categorical data. The same analysis was conducted in [27].

To be as much as possible conservative (because of the sample size and mostly non-normality of the data), a non-parametric test has been used to test the hypotheses related to differences in the subjects' *time* and *efficiency* in performing comprehension and change tasks. This choice is in agreement with the suggestions by Motulsky [22] in Chapter 37. The unpaired analysis—i.e., an analysis of all data grouped by different treatments of the main factor—is performed using the Mann-Whitney, (one- and two- tailed) test [29]. 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 this test allows for checking the presence of significant differences, it does not provide any information about the magnitude of such a difference. This is particularly relevant in our study, since we are interested in investigating to what extent the use of obfuscation reduces the likelihood of completing an attack and increases the time needed for an attack. To this aim, two kinds of effect size measures have been used, the *odds ratio* and the non-parametric Cliff's delta (d) effect size [18].

The odds ratio is a measure of effect size that can be used for dichotomous categorical data. An odds [29] indicates how likely it 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:

$$OR = \frac{p/(1-p)}{q/(1-q)} \tag{2}$$

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.

For independent samples, Cliff's delta provides an indication of the extent to which two (ordered) data sets overlap, i.e., it is based on the same principles of the Mann-Whitney test. For dependent samples, it is defined as the probability that a randomly selected member M_1 of one sample has a higher response than a randomly selected member M_2 of the second sample, minus the reverse probability. Formally:

$$d = Pr(\mathrm{M}_1^i > \mathrm{M}_2^j) - Pr(\mathrm{M}_2^j > \mathrm{M}_1^i)$$

A sample estimate of this parameter can be obtained by enumerating the number of occurrences of a sample one member having a higher response value than a sample two member, and the number of occurrences of the reverse. This gives the sample statistic:

$$d = \frac{\left|\mathrm{M}_1^i > \mathrm{M}_2^j\right| - \left|\mathrm{M}_2^j > \mathrm{M}_1^i\right|}{|\mathrm{M}_1|\,|\mathrm{M}_2|}$$

Cliff's Delta ranges in the interval $[-1 \ldots 1]$. It is equal to $+1$ when all values of one group are higher than the values of the other group and -1 when reverse is true. Two overlapping distributions would have a Cliff's Delta equal to zero. The effect size is considered small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$ and large for $d \geq 0.474$ [7].

To provide a picture of what a worst case scenario (fastest attack) could look like we compute, for each experiment and for each system used in the experiment, the lowest times (expressed in minutes) achieved in correctly answering comprehension questions (T1, T2) and performing change tasks (T3, T4). We compare the difference between the obfuscated and clear cases to the pooled standard deviation (as for the Cohen d). We deem relevant the differences that are $\geq \sigma$. A case of relevance difference is also when no correct answers are delivered with one treatment, while correct answers are delivered with the other treatment. Although we cannot claim statistical significance and therefore no specific hypothesis was formulated, we believe this measure provides useful insights.

The analysis of co-factors, as well as the hypothetical effect of confounding factors such as System and Lab, is performed using permutations test [2], and interactions are visualized using interaction plots. The permutation test is a non-parametric alternative to the two-way Analysis of Variance (ANOVA); differently from ANOVA, it does not require data to be normally distributed. The general idea behind such a test is that the data distributions are built and compared by computing all possible values of the test statistic while rearranging the labels (representing the various factors being considered) of the data points. We used the implementation available in the *lmPerm* R package. We have set the number of iterations of the permutation test procedure to 500,000. Since the permutation tests samples permutations of combination of factor levels, multiple runs of the test may produce different results. We made sure to choose a high number of iterations such that results did not vary over multiple executions of the procedure.

The permutation tests relate the dependent variable—i.e., efficiency in the comprehension and in the change task—with:

1. the main factor Treatments, i.e., the kind of obfuscation applied (or no obfuscation);
2. the considered co-factors, i.e., subjects' Experience, System considered in the study, and Lab in which the task was performed;
3. the interaction between the main factor and the co-factors;
4. the interaction between co-factors themselves.

Note that we only considered two-way interactions. Three- or four-ways interactions cannot be applied due to the limited number of data points, and also would not be very meaningful to be interpreted.

To analyze the effect of the *learning across subsequent tasks*, we used a repeated measures permutation test, which is, again, the non-parametric alternative to the Repeated Measures ANOVA. Specifically, this test allows to distinguish the *between subjects* variance, due to the application of different treatments to different subjects, from the *within subjects* variance, due to (i) different treatments received by each subject due to the experimental design, (ii) the ordering and possible different difficulty of the questions being asked, and (iii) the interaction between these two factors.

Regarding the analysis of survey questionnaires, we evaluate questions related to objectives clarity, availability of enough time and general difficulties subjects might have encountered (Q1-Q4, Q8, Q10) by verifying that the answers are either Strongly agree (1) or Agree (2). Similarly to [25], we test medians, using a one-tailed Mann-Whitney test for the null hypothesis $\widetilde{Qx} \geq 3$, where 3 corresponds to "Undecided", and $\widetilde{Qx}$ is the median for question Qx. A similar analysis is performed, only for subjects receiving obfuscated code, for questions related to the use made of the debugger (Q9), the difficulty in comprehending obfuscated code (Q17) and the usefulness of executing the system to understand it when the code is obfuscated (Q18).

For the questions related to the ability of subjects to perform comprehension, feature location, and change tasks (Q5, Q6, Q7), answers of subjects receiving the first treatment TR_1 (i.e., clear code) were compared with answers of subjects receiving the second treatment TR_2 (i.e., obfuscated code). In this case a two-tailed Mann-Whitey test is used for the null hypothesis $\widetilde{Q_{TR_1}} = \widetilde{Q_{TR_2}}$. A similar comparison is also performed for questions concerning the usefulness of debugging (Q11) and automatic renaming (Q12), and for questions concerning the number of executions (Q13), debugging runs (Q14) and time spent looking at the code (Q15) and running the system (Q16).

In all the statistical tests performed, we considered a 95% significance level, i.e., we accept a 5% probability of committing a Type I error.

3.7 Threats to Validity

We identified the main threats to the validity that can affect our results [33]: construct, internal, conclusion, and external validity threats.

Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the capability of a subject to perform an attack. The tasks were chosen to be as representative as possible of realistic attacks. Also, the measurements we conceived—comprehension questions with one possible answer and test cases to assess code correctness—are as objective as possible. Clearly, the ability to understand the questions we asked might not fully reflect the comprehension achieved by the subject for that particular task. Also, the test cases we used cover only the scenario we asked to modify in the attack task. Alternative scenarios are not tested, as well as code not directly involved in the scenario that might have been impacted by the change.

Internal validity threats concern additional factors that may affect an independent variable. They can be due to learning and fatigue effects. Since the common design envisages a sequence of two labs in which, although with different treatments, the same type of task is required, it is possible to observe a learning effect; as a consequence a subject's performance could improve from the first to the second lab. The chosen design should mitigate the possible confounding effect of learning on the main factor effect. To limit the fatigue effect, we introduced a break between the two tasks. Moreover, subjects were not aware of the study hypotheses, and were told they would not be evaluated on the performance observed during the experiment.

Conclusion validity concerns the relationship between the treatment and the outcome. To analyze correctness, we opted for the Fisher's exact test, more accurate than the χ^2 test for small sample sizes. On the contrary, for analyzing time and efficiency we selected a non-parametric test (i.e., Mann-Whitney), because it is very robust and sensitive [22] (see chapter 37). Similarly, the analysis of co-factor has been performed using permutation test, which is a non-parametric alternative to ANOVA and does not require data to be normally distributed as ANOVA does. Survey questionnaires, mainly intended to get qualitative insights, were designed using standard structure and scales [23].

External validity concerns the generalization of the findings. First, only two types of obfuscation—identifier renaming and opaque predicates— were considered. While for identifier renaming the strength of the obfuscation does not change across different implementations, strength of opaque predicates may vary, depending on the complexity of predicates and on of what code is generated in the dead branches. Then, although we considered two different distributed systems belonging to different domains and having a different complexity, further studies with different systems are desirable. Last, but not least, the studies were

performed in academic environments. Although for this type of experiment (hacker attack) it is not interesting to experiment with industrial developers, we are aware that the expertise of students could be far from that of hackers. However, hackers are not easily available and the only pragmatic possibility is resorting to students. Moreover, this threat was at least mitigated (i) by considering students with different level of experience, (ii) by analyzing the worst case scenario, and (iii) by performing a co-factor analysis by experience. All in all, many hackers are not that different from best students (high Experience subjects/worst case scenarios in our experiments). Clearly, further studies with larger groups of objects, more demanding tasks, and more experienced participants are needed to confirm or contradict the results from this study.

4 Results

This section reports the results of the five experiments, with the aim of testing the hypotheses formulated in Section 3.3. Working data sets are available for replication purposes[9].

4.1 Analysis of correctness

Table 5 Number of correct tasks and results of Fisher's test. Significant p-values are shown in bold face.

Exp	Hyp.	Treat. 1	Treat. 2	Clear code		Identifier renaming		Opaque predicates		Analysis	
				Correct	Wrong	Correct	Wrong	Correct	Wrong	p-value	OR
I	H_1	Clear	IR	11	7	8	12	–	–	0.16	0.43
II	H_1	Clear	IR	23	15	27	17	–	–	0.62	1.04
III	H_3	Clear	OP	20	12	–	–	19	9	0.76	1.26
IV	H_5	IR	OP	–	–	8	14	13	12	0.38	1.87
V	H_5	IR	OP	–	–	12	12	12	12	1.00	1.00

(a) Comprehension

Exp	Hyp.	Treat. 1	Treat. 2	Clear code		Identifier renaming		Opaque predicates		Analysis	
				Correct	Wrong	Correct	Wrong	Correct	Wrong	p-value	OR
I	H_2	Clear	IR	15	3	8	12	–	–	**0.01**	0.14
II	H_2	Clear	IR	33	5	24	16	–	–	**0.01**	0.23
III	H_4	Clear	OP	21	11	–	–	22	6	0.92	1.90
IV	H_6	IR	OP	–	–	6	2	9	3	1.00	1.00
V	H_6	IR	OP	–	–	0	17	2	19	0.49	Inf

(b) Change

Table 5 reports the analysis of correctness of the tasks performed by our subjects, each experiment in a different line. The table reports the hypothesis tested and the treatments considered in each experiment. Then, for each treatment, the number of correct and wrong tasks (both comprehension and change tasks), the p-value resulting from the Fisher's test and the effect size, computed as the odds ratio (an odds ratio < 1 indicates that the chances of success are higher with the first treatment than with the second one). Significant p-values $<$0.05 are shown in bold face.

No statistical significance can be observed for comprehension tasks, suggesting that obfuscation does not impact the likelihood of comprehension tasks to be performed correctly.

[9] http://selab.fbk.eu/ceccato/replication_packages/id_renaming_vs_opaque_predicates_package.tgz

However, for change tasks, statistical significance (p-value shown in bold face) can be observed for Exp I and Exp II with an OR < 1 (0.14 in Exp I and 0.23 Exp II), indicating that the chances of obtaining correct comprehension are about 7 and 4 times higher with clear code than with obfuscated code respectively. No statistical significance can be observed for Exp III, suggesting that opaque predicates obfuscation offers a limited level of protection and that it is not effective in making change tasks harder. At a first glance, data seems to be not consistent, since the direct comparison between the two obfuscation techniques (Exp IV and Exp V) does not reach statistical significance, suggesting that the level of protection achieved by them is similar, while identifier renaming has a statistically significant effect for change tasks, when compared to clear code, while opaque predicates does not have any statistically significant effect. However, it should be noticed that transitivity does not hold for statistical significance. Subsequent analysis on *time* and *efficiency* will provide more insight in this point.

Overall, hypothesis H_{02} on identifier renaming can be rejected using the Fisher's test on Exp I and Exp II. Therefore, we can formulate the following alternative hypothesis:

– H_{A2}: Identifier renaming obfuscation *decreases* the capability of an attacker to perform correct change tasks.

4.2 Analysis of time

Fig. 2 shows boxplots of the time required to deliver correct answers (wrong answers are discarded for this analysis) on all the five experiments, divided in (a) comprehension tasks and (b) change tasks. From the graphs we can observe that obfuscated code (either with identifier renaming and opaque predicates) appears to require more time for a correct comprehension and for elaborating a correct attack (Exp I, II and III). Strangely enough, the trend observed for the direct comparison of the two obfuscation (Exp IV and V) has alternating directions. In fact, comprehension tasks require more time with identifier renaming, while change tasks require more time with opaque predicates. As it can be noticed, there is no boxplot for identifier renaming in Exp V, as in such a case nobody was able to correctly complete any change task when this obfuscation was deployed.

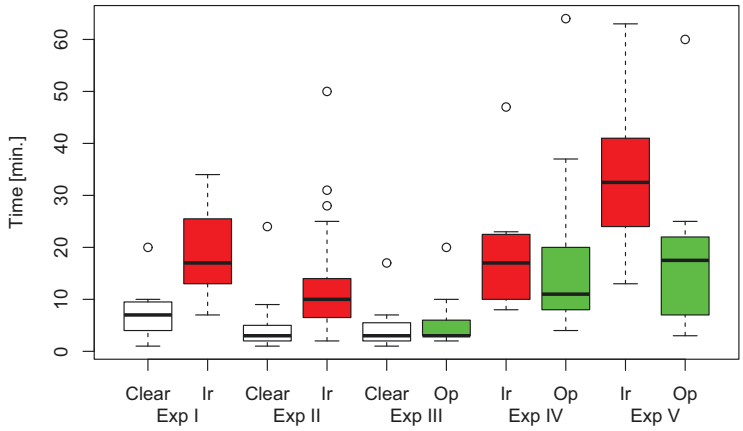
Table 6 Unpaired analysis of time to attack.

Exp	Hyp.	Treat. 1					Treat. 2					Analysis	
		name	N	mean	median	σ	name	N	mean	median	σ	p-value	Cliff's d
I	H_1	clear	11	7.4	7.0	5.3	ir	8	19.0	17.0	8.8	**<0.01**	-0.78
II	H_1	clear	23	4.8	3.0	4.8	ir	27	12.5	10.0	10.5	**<0.01**	-0.65
III	H_3	clear	20	4.2	3.0	3.5	op	19	5.2	3.0	4.3	0.18	-0.17
IV	H_5	ir	8	19.3	17.0	12.5	op	13	17.8	11.0	17.1	0.43	0.22
V	H_5	ir	12	32.9	32.5	14.1	op	12	17.9	17.5	15.3	**0.01**	

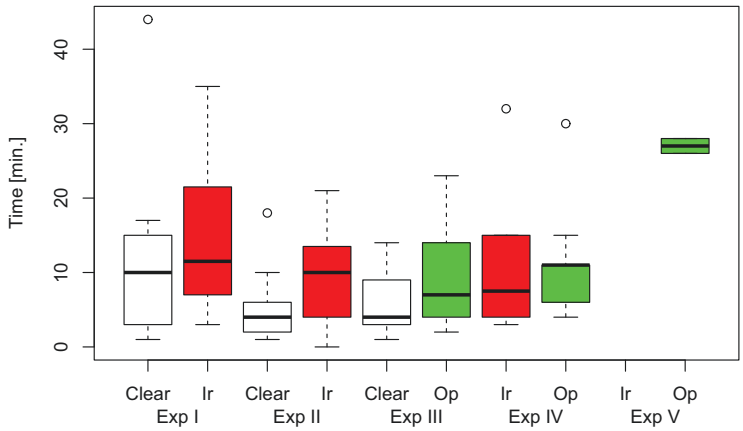
(a) Comprehension

Exp	Hyp.	Treat. 1					Treat. 2					Analysis	
		name	N	mean	median	σ	name	N	mean	median	σ	p-value	Cliff's d
I	H_2	clear	14	11.4	10.0	11.0	ir	8	14.8	11.5	10.8	0.18	-0.25
II	H_2	clear	33	4.5	4.0	3.4	ir	24	9.9	10.0	6.4	**<0.01**	-0.52
III	H_4	clear	21	5.6	4.0	4.0	op	22	8.6	7.0	6.0	**0.02**	-0.35
IV	H_6	ir	6	11.5	7.5	11.1	op	9	11.3	11.0	7.8	0.68	-0.15

(b) Change



(a) Comprehension



(b) Change

Fig. 2 Boxplots of time to attack. "Clear" (white) = clear code; "Ir" (red) = code obfuscated with identifier renaming; "Op" (green) = code obfuscated with opaque predicates.

Table 6 reports descriptive statistics and the unpaired analysis of the time required to deliver correct answers. For each experiment, the table reports the tested hypothesis, the two alternative treatments, the number of subjects that participated, mean, median and standard deviation of the time needed to deliver correct answers and elaborate correct change tasks. P-values of the Mann-Whitney unpaired test and the Cliff's *d* effect size are reported for

the experimental data (negative effect sizes indicate that values observed with the second treatment are higher than those observed with the first treatment). No analysis is reported for change tasks on Exp V, because only two subjects performed a correct change task, and both of them under the same treatment.

The amount of time required to correctly answer comprehension tasks is significantly longer when working with identifier renaming than when working with clear code (Exp I and II) and with opaque predicates (Exp V) with a large effect size ($d \geq 0.47$). Statistical significance is not reached on Exp III when comparing opaque predicates with clear code. This suggests that code obfuscated with identifier renaming requires a longer amount of time to be understood, but this effect is not observed when the code is obfuscated with opaque predicates.

When facing change tasks, the time required to attack an obfuscated program is significantly longer than when changing clear code, when identifier renaming is used (Exp II) the effect size is large ($d \geq 0.47$), when opaque predicates is used (Exp III) the effect size is medium (d=0.35). No statistical significance can be observed in the direct comparison (Exp IV and Exp V).

For Exp I, we can reject hypothesis H_{01} on identifier renaming, while we cannot reject H_{02}. Instead, for Exp II, we can reject both H_{01} and H_{02}. Exp III allows us to reject H_{04} on opaque predicates In Exp IV we did not observe any significant difference between identifier renaming and opaque predicates. Instead, in Exp V we could reject H_{05}, concerning the comparison between identifier renaming and opaque predicates obfuscations on comprehension tasks.

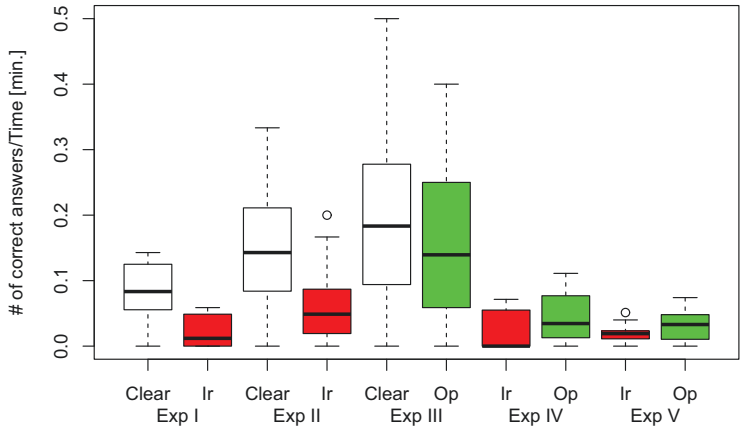
Therefore, we can accept the following alternative hypotheses (in parentheses we report the experiments for which such conclusions are valid):

- H_{A1}: identifier renaming obfuscation significantly *decreases* the *time* required to perform a correct *comprehension task* (Exp I and Exp II).
- H_{A2}: identifier renaming obfuscation significantly *decreases* the *time* required to perform a correct *change task* (Exp II only).
- H_{A4}: opaque predicates obfuscation significantly *decreases* the *time* required to perform a correct *change task* (Exp III).
- H_{05}: identifier renaming is significantly *more effective* than opaque predicates in increasing the *time* required to perform a correct *comprehension task* (Exp V only).

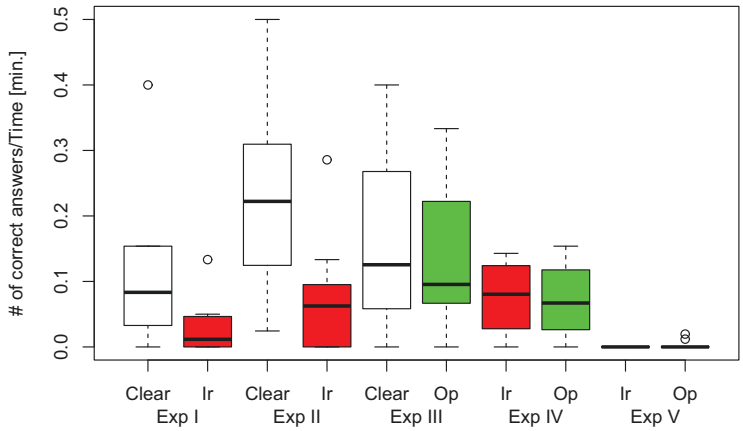
4.3 Analysis of efficiency

Fig. 3 shows boxplots of the number of correct answers per minute (efficiency) for all the five experiments, divided by comprehension and change tasks (respectively (a) and (b) in Fig. 3). As also the figure highlights, both comprehension and change tasks appear to be performed more efficiently on clear code than on code obfuscated with identifier renaming (Exp I and Exp II). Reduced efficiency of obfuscation is not so evident when we compare clear code with code obfuscated with opaque predicates (Exp III). The direct comparison of the two obfuscations (Exp IV and Exp V) partially confirms this result. In fact, comprehension tasks are conducted more efficiently on code obfuscated with opaque predicates, while change tasks report very similar efficiency. In particular, in Exp V (bachelor students) only very few subjects could correctly deliver change tasks with either obfuscation techniques.

Table 7 reports the descriptive statistics and the unpaired analysis of attack efficiency, each experiment in a different line. The table reports the hypotheses tested in each experiment, the two treatments adopted, the number of subjects who participated to it, mean,



(a) Comprehension



(b) Change

Fig. 3 Boxplots of attack efficiency. "Clear" (white) = clear code; "Ir" (red) = code obfuscated with identifier renaming; "Op" (green) = code obfuscated with opaque predicates.

median and standard deviation of efficiency. Then p-values computed by the Mann-Whitney unpaired test and the Cliff *d* effect size are reported.

Both on comprehension and on change tasks, subjects working on clear code outperformed subjects working on code obfuscated with identifier renaming (Exp I and Exp II) in

Table 7 Unpaired analysis of Efficiency of attacks.

Exp	Hyp.	Treat. 1					Treat. 2					Analysis	
		name	N	mean	median	σ	name	N	mean	median	σ	p-value	Cliff's d
I	H_1	Clear	9	0.08	0.08	0.05	Ir	10	0.02	0.01	0.02	<**0.01**	0.77
II	H_1	Clear	19	0.15	0.14	0.10	Ir	22	0.06	0.05	0.06	<**0.01**	0.50
III	H_3	Clear	16	0.19	0.18	0.14	Op	14	0.16	0.14	0.12	0.33	0.10
IV	H_5	Ir	12	0.02	0.00	0.03	Op	13	0.05	0.03	0.04	0.11	-0.37
V	H_5	Ir	12	0.02	0.02	0.01	Op	12	0.03	0.03	0.02	0.16	-0.35

(a) Comprehension

Exp	Hyp.	Treat. 1					Treat. 2					Analysis	
		name	N	mean	median	σ	name	N	mean	median	σ	p-value	Cliff's d
I	H_2	Clear	9	0.11	0.08	0.12	Ir	10	0.03	0.01	0.04	**0.05**	0.77
II	H_2	Clear	19	0.24	0.22	0.17	Ir	22	0.07	0.06	0.07	<**0.01**	0.50
III	H_4	Clear	16	0.18	0.13	0.18	Op	14	0.13	0.10	0.10	0.31	0.10
IV	H_6	Ir	12	0.08	0.08	0.06	Op	13	0.07	0.07	0.06	1.00	-0.37
V	H_6	Ir	12	0.00	0.00	0.00	Op	12	0.00	0.00	0.01	0.19	-0.35

(b) Change

a statistically significant way, with a large effect size ($d \geq 0.47$). This suggests that the first obfuscation makes the code substantially harder to understand and change.

Then, when considering opaque predicates, subjects working with obfuscated code exhibited a performance similar to subjects working on clear code (Exp III), suggesting that this second obfuscation offers a very limited protection against attacks. However, when comparing directly the two obfuscations (Exp IV and Exp V), the difference in subjects' performance is not statistically significant.

Overall, hypotheses H_{01} and H_{02} on identifier renaming can be rejected using data from Exp I and Exp II. Hypotheses H_{03} and H_{04} on opaque predicates cannot be rejected on Exp III. Therefore, we can formulate the following alternative hypotheses:

- H_{A1}: identifier renaming obfuscation significantly *decreases* the *efficiency* of an attacker performing *comprehension tasks*.
- H_{A2}: identifier renaming obfuscation significantly *decreases* the *efficiency* of an attacker performing *change tasks*.

4.4 Analysis of worst case scenario

Depending on their security requirements, applications protected by obfuscation may suffer the problem known as *break once run everywhere*. According to this concern, the first attacker able to break the obfuscation may share the solution or distribute a "crack", i.e., a small program encoding the attacker's knowledge to automatically bypass the protection. Thus, once the obfuscation is broken by the fastest attacker, all instances of the application should be considered insecure.

To measure the protection offered against the *break once run everywhere* pattern, we compare the amount of time taken by the fastest subject to complete comprehension and change tasks. For each experiment, we identify the shortest time (best attack) taken by subjects to complete successful tasks, when working on decompiled clear code and on code protected with different obfuscations. Such best time cases correspond to the worst cases from the obfuscator point of view.

Table 8 reports the shortest time (expressed in minutes) taken to successfully complete tasks when different protections are deployed (different obfuscations or clear code) when working on different object applications. The table also reports the pooled standard deviation

Table 8 Lowest times for successful attacks.

		ChatClient				CarRace			
Exp	Treatment	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4
I	clear	1	20	3	15	2	7	3	1
	ir	25		18	9	15	7	12	3
	σ_{pooled}	**4.2**		29.0	6.3	**6.6**	3.7	12.0	4.3
II	clear	1	3	2	3	2	2	1	1
	ir	5	2	4	1	4	2	10	0
	σ_{pooled}	10.9	7.9	5.2	3.6	3	5.6	**4.7**	3.6
III	clear	1	5	2		2	1	2	1
	op	2	5	2	5	2	3	3	3
	σ_{pooled}	3.0	4.8	6.2		1.3	5.1	4.5	5.6
IV	ir	8	22	11	3	16	10	15	4
	op	5		11	5	4	4	9	4
	σ_{pooled}	18.9				**9.1**	13.5	8.8	2.6
V	ir	30	23			26	13		
	op	10	18			21	3		26
	σ_{pooled}	**5.5**	9.3			19.2	**7.1**		

of the time for successful attacks. The relevant cases (pooled standard deviations smaller than the difference of lowest times) are highlighted in boldface. Relevant cases are also those where subject could deliver no correct answers when working with one treatment, but could when working with the other treatment.

We observe three relevant differences when comparing identifier renaming with clear code, two in Exp I and one in Exp II. For the ChatClient system on Exp I the fastest attack on code obfuscated with identifier renaming in T1 takes **25 times** longer than on clear code, while no one could complete correctly T2 on obfuscated code, compared to the 20 minutes required on clear code. For the CarRace system, on Exp I the fastest attack on obfuscated code in T1 takes more than **7 times** longer than on clear code, while on Exp II the fastest attack for T3 on obfuscated code takes **10 times** longer than on clear code.

Considering the comparison between code obfuscated with opaque predicates and clear code, in Exp III, only one relevant case can be observed. No one could accomplish task T4 on the clear code of the system ChatClient.

Considering the direct comparison of the two obfuscations (Exp IV and V), there are five relevant differences. On the system ChatClient, no subject performed correctly task T2 when working with opaque predicates in Exp IV and, in Exp V, the amount of time required to correctly perform task T1 with identifier renaming was **3 times** longer than the same task on opaque predicates. On CarRace, the fastest subject who worked with identifier renaming on Task T1 in Exp IV took **4 times** longer than the fastest subject who had opaque predicates. A similar ratio applies to Exp V, task T2. On Exp V, the only case for which we observe a change task (T3 and T4) correctly performed is for a subject who worked on code obfuscated with opaque predicates (T4 on CarRace).

4.5 Analysis of co-factors

This section reports the analysis of co-factors that could have influenced the results of our experiments, with respect to the efficiency of completing comprehension and change tasks.

Results of the permutation test for the comprehension tasks are reported in Table 9. Specifically, results indicate that:

Table 9 Comprehension task: Analysis of the influence of co-factors, of their interaction with the main factor and between themselves.

	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	0.19	0.09	500,000	<**0.01**
Experience	2	0.04	0.02	500,000	0.05
Treatment:Experience	2	0.02	0.01	500,000	0.20
System	1	0.02	0.02	500,000	0.10
Treatment:System	2	0.00	0.00	308,611	0.95
Experience:System	2	0.01	0.00	500,000	0.53
Lab	1	0.03	0.03	500,000	**0.03**
Treatment:Lab	2	0.04	0.02	500,000	0.05
Experience:Lab	2	0.00	0.00	132,413	0.98
System:Lab	1	0.01	0.01	500,000	0.33
Residuals	121	0.80	0.01		

– The subjects' Experience has a marginal effect on the code comprehension. However, there is no significant interaction with the Treatment, i.e., more experienced subjects performed better than less experienced ones, regardless of the level of obfuscation;
– The characteristics of the objects (System) does not have any significant effect, nor any interaction with the Treatment or with the other co-factors;
– The Lab has a significant effect, as well as a marginal interaction with the Treatment. Looking at the results more in detail (see also Fig. 4), we can infer that the improved efficiency due to learning is more relevant on clear code than on obfuscated code. In other words, previous experience in performing attack tasks is valuable when facing clear code, but it is almost irrelevant when working on obfuscated code.

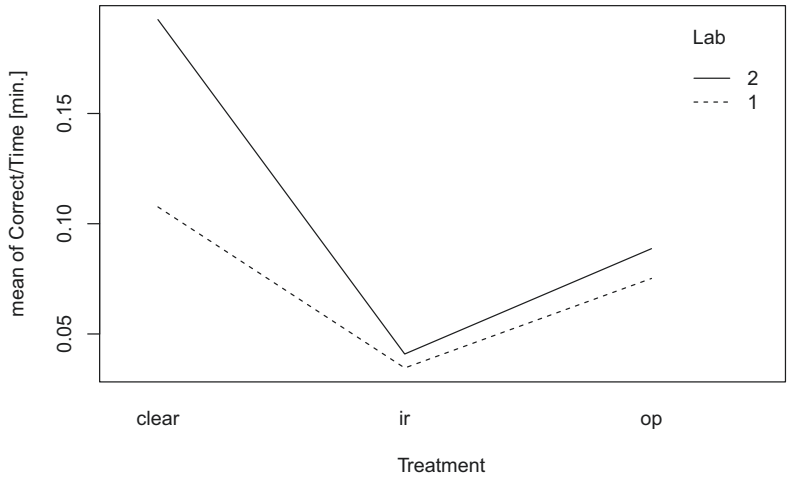


Fig. 4 Interaction plot of Treatment & Lab (Comprehension tasks).

Table 10 reports permutation test results for change tasks. Results indicate that:

– The subjects' Experience has a significant effect, although it does not interact with the main factor and with other factors;

Table 10 Change task: Analysis of the influence of co-factors, of their interaction with the main factor and between themselves.

	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	0.06	0.03	500,000	0.05
Experience	2	0.21	0.10	500,000	<**0.01**
Treatment:Experience	2	0.05	0.02	500,000	0.10
System	1	0.14	0.14	500,000	<**0.01**
Treatment:System	2	0.17	0.08	500,000	<**0.01**
Experience:System	2	0.01	0.00	500,000	0.62
Lab	1	0.02	0.02	500,000	0.14
Treatment:Lab	2	0.02	0.01	500,000	0.27
Experience:Lab	2	0.01	0.01	500,000	0.51
System:Lab	1	0.01	0.01	500,000	0.24
Residuals	95	0.89	0.01		

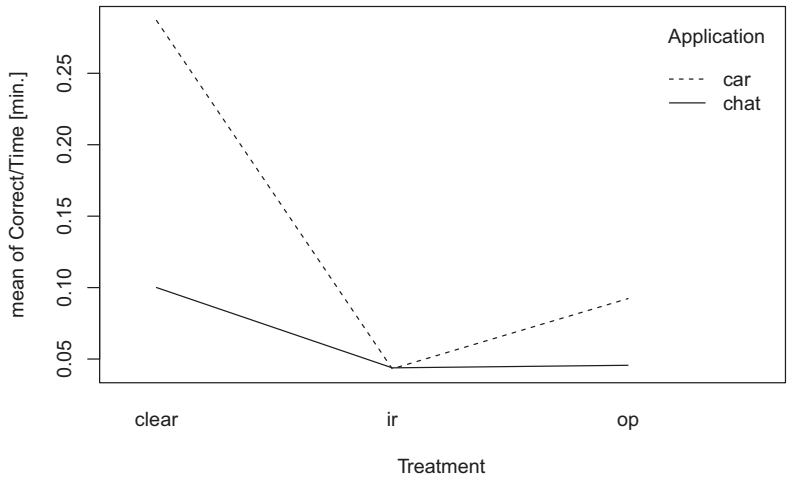


Fig. 5 Interaction plots of Treatment & System (change tasks).

- There is a significant effect of the System on which the task was performed (i.e., CarRace or ChatClient) on the efficiency of the change task. This result can be interpreted by looking at the interaction plot of Fig. 5: although the CarRace system is always easier to attack than the ChatClient system, the difference is reduced when obfuscating the code (both with identifier renaming and opaque predicates).
- The Lab in which the task was performed does not have any significant effect, nor it interacts with the main factor or with other factors.

Finally, we analyzed the learning effect across questions with the repeated measures Permutation Test for comprehension and change tasks respectively (see Table 11 and Table 12). The test does not report any *within subject* significant effect of the Question, nor any interaction between Question and Treatment. This suggests that the difficulty of the different questions, as well as their ordering, does not influence the experimental results.

Table 11 Repeated measures Permutation Test of Efficiency by Treatment & Question (comprehension tasks).

Between Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	0.30	0.15	500,000	$<$ **0.001**
Residuals	66	0.60	0.01		
Within Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Question	1	0.00	0.00	51	1.00
Treatment:Question	2	0.00	0.00	51	1.00
Residuals	66	0.00	0.00		

(a) ChatClient

Between Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	0.30	0.15	500,000	$<$ **0.001**
Residuals	67	1.33	0.02		
Within Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Question	1	0.00	0.00	51	1.00
Treatment:Question	2	0.00	0.00	51	1.00
Residuals	67	0.00	0.00		

(b) CarRace

Table 12 Repeated measures Permutation Test of Efficiency by Treatment & Question (change tasks).

Between Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	0.09	0.05	500,000	**0.02**
Residuals	51	0.62	0.01		
Within Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Question	1	0.00	0.00	51	1.00
Treatment:Question	2	0.00	0.00	51	1.00
Residuals	51	0.00	0.00		

(a) ChatClient

Between Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Treatment	2	1.31	0.65	500,000	$<$**0.001**
Residuals	56	1.82	0.03		
Within Subjects					
	Df	R Sum Sq	R Mean Sq	Iter	Pr(Prob)
Question	1	0.00	0.00	51	1.00
Treatment:Question	2	0.00	0.00	51	1.00
Residuals	56	0.00	0.00		

(b) CarRace

4.6 Analysis of post-experiment survey questionnaire

The post-experiment survey questionnaire (see Table 4) is aimed at both gaining insights about the subjects' behavior during the experiment and finding justifications for the quantitative results.

Table 13 Survey questionnaire analysis: objectives clarity and problems encountered with time/settings and Obfuscation-specific questions (Mann-Whitney for median(Q_x)$\geq$3). Medians $\widetilde{Qx}$ on the left hand side of the column and p-values on the right hand side.

Exp	Q1		Q2		Q3		Q4		Q8		Q10		Q17		Q18	
	$\widetilde{Q1}$	p-value	$\widetilde{Q2}$	p-value	$\widetilde{Q3}$	p-value	$\widetilde{Q4}$	p-value	$\widetilde{Q8}$	p-value	$\widetilde{Q10}$	p-value	$\widetilde{Q17}$	p-value	$\widetilde{Q18}$	p-value
All	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**
I	2	0.08	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	3	0.47	2	**<0.01**	2	**0.01**
II	1	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	1	**<0.01**
III	1	**<0.01**	2	**<0.01**	1	**<0.01**	1	**<0.01**	1	**<0.01**	2	**<0.01**	2	**0.02**	2	**0.01**
IV	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	2	**<0.01**	3	0.30	2	**<0.01**	2	**<0.01**
V	2	**0.03**	2	**0.02**	2	**<0.01**	2	**<0.01**	2	**0.01**	3	0.31	1	**<0.01**	2	**<0.01**

Table 14 Effect of treatment on comprehension and maintenance (Mann-Whitney for median(Q_{t1})=median(Q_{t2})).

Exp	Treatments		Q5	Q6	Q7	Q9	Q11	Q12	Q13	Q14	Q15	Q16
I	Clear	Ir	0.52	**0.04**	0.20	0.38	0.73	0.28	**0.04**	0.28	1.00	0.50
II	Clear	Ir	**<0.01**	**<0.01**	**<0.01**	0.25	0.34	**0.04**	0.95	**0.01**	0.22	0.24
III	Clear	Op	0.66	1.00	0.57	0.19	0.76	0.22	0.76	0.44	0.28	0.93
IV	Ir	Op	1.00	0.23	1.00	0.69	0.16	0.91	0.43	0.66	0.93	0.31
V	Ir	Op	0.29	**0.05**	0.18	0.80	0.89	0.46	0.51	0.70	0.97	0.20

Questions **Q1**, **Q2**, **Q3** and **Q4** are intended to perform a validation of the clarity of the experimental tasks and objectives, while **Q8** and **Q10** aim at check possible problems occurred with the experimental settings. Questions **Q17** and **Q18** are intended to identify possible problems encountered when working on obfuscated code.

No general problem emerged from the analysis of survey questionnaire questions (**Q1**, **Q2**, **Q3**, **Q4**, **Q8**, **Q10**) when considering answers from all the experiments together (first line of Table 13). Only a few specific problems emerged on some experiments, related to the overall subjects' ability to perform the tasks in the allotted time and related to the clarity of the lab objectives. Subjects of Exp I experienced problems regarding the time needed to perform the task (**Q1**), and to use the debugger (**Q10**). Debugger was a problem also in Exp IV and Exp V. No particular problem occurred in Exp II (more experienced subjects) and Exp III.

Subjects from all the experiments agreed (p-value $<$**0.01**) that the obfuscated code was more difficult to understand (**Q17**), and that system execution is necessary for understanding the behavior of the code (**Q18**), as a complement to static code analysis.

After the assessment of the experimental settings, we compared the answers provided by subjects, when using clear and obfuscated code, or code obfuscated with two different obfuscation techniques (see Table 14), on the difficulties encountered in code comprehension (**Q5**), location of the feature to be understood/changed (**Q6**), and in performing the change task (**Q7**). In Exp I, subjects felt that identifier renaming makes feature location more difficult (p-value=0.04), while there is no difference for code comprehension (p-value=0.52) and change (p-value=0.20) tasks. In Exp II, subjects felt that identifier renaming makes all three activities—comprehension, feature location and change—more difficult (p-value < 0.01 in all cases). In Exp III, opaque predicates did not cause major problems to comprehension, feature location and change when compared to the clear code (p-value > 0.05 in all cases). In the direct comparison of the two obfuscations, identifier renaming is reported as more problematic than opaque predicates with respect to identification of the features to change only in Exp V, but not in Exp IV. This partially confirms the quantitative analysis.

We also investigated the perceived usefulness of the tools available to the subjects, i.e., the use (**Q9**) and usefulness of the debugger (**Q11**) and the renaming facility provided by Eclipse (**Q12**). In general, subjects did not report a different usage of these facilities when facing different treatments. The only reported difference is in Exp II (more expert subjects), where the renaming facilities were used more extensively when working with code obfuscated with identifier renaming.

When performing comprehension and change tasks, we investigated whether there was a variation—between subjects with different treatments—in the number of system executions (**Q13**) and executions in debugging mode (**Q14**) reported by subjects, the percentage of time spent looking at the code (**Q15**), and running the system (**Q16**). A significant difference was found in Exp I, where subjects felt they needed to execute the system more times (**Q13**, p-value=0.04) when it was obfuscated with identifier renaming (4 to 10 executions) than when it was in clear (2 to 4 executions). While in Exp II more executions were performed in debugging mode (**Q14**, p-value=0.01) when working on obfuscated code (2 to 4 executions) with respect to clear code (just 1 execution), although subjects said they used the debugger for obfuscated code as often as for clear code (**Q9**, p-value=0.38 in Exp I and 0.25 in Exp II). Results suggest that in Exp I subjects used system executions as a way to better understand obfuscated systems, but they did not use the debugger, differently from the subjects of Exp II, since they felt debugging difficult to perform (as reported in the answers to question **Q10**).

Therefore, on the results of the post-experiment survey questionnaire, participants reported that:

- Obfuscated code was difficult to understand;
- They had to execute the code for understanding the behavior of obfuscated code, especially when identifier renaming was used;
- For some of them, attacking code obfuscated with identifier renaming was difficult, mainly because this obfuscation made features more difficult to locate. However, after having located features, only few participants had problems in understanding and in changing them;
- In code obfuscated with opaque predicates, features were not particularly hard to locate, understand and change;
- Code obfuscated with identifier renaming was more difficult to attack that code obfuscated with opaque predicates, because in the first case features were more difficult to identify;
- The debugger was not useful to attack obfuscated code, mainly because the debugger was difficult to use on obfuscated code;
- Renaming facilities were useful just in few cases, and just to attack code obfuscated with identifier renaming.

5 Discussion

The quantitative results reported in Section 4 allow us first to draw conclusions for each individual technique—identifier renaming and opaque predicates—and then to outline some general observations.

5.1 Identifier Renaming

- *Identifier Renaming obfuscation represents an effective protection technique*: when the source code is obfuscated with Identifier Renaming, the capability of an attacker to understand the code decreases in terms of time and efficiency. The capability to change the code decreases in terms of accuracy, time and efficiency (See the analysis of correctness, time and efficiency). By comparing the average time spent to correctly change clear and obfuscated code we observe that, on average, an attack on the code obfuscated with Identifier Renaming takes 2 times longer than on the clear code (see the analysis of time). In the worst case, it takes 10-25 times more than the clear code (see the analysis of worst case).
- *Renaming facilities can weaken Identifier Renaming*: once the intended purpose of an identifier is recovered, the attacker can change the till-that-point meaningless identifier into a meaningful one, using renaming facilities (e.g., those provided by a development environment). However, to properly use renaming facilities, some level of experience is required. We observed that inexperienced attackers prefer not to change the code, and work on the obfuscated code directly (see post-questionnaire analysis, Q12).

5.2 Opaque Predicates

- *Opaque Predicates obfuscation offers a limited protection*: when the code is obfuscated with Opaque Predicates, the capability of an attacker to change the source code is reduced in terms of the required time to perform the attack; no significant change is observed in terms of correctness and efficiency of the attack. Code obfuscated with Opaque Predicates requires 20%-50% more time to be attacked than clear code (see analysis of time and worst case).
- *Opaque Predicates obfuscation does not make features difficult to locate*: Opaque predicate obfuscation obstructs code comprehension by complicating the control flow. However, perfect comprehension of the control flow is not required to elaborate an attack; a limited knowledge of an important portion of code is sufficient. Indeed, subjects reported that when the code is obfuscated with Opaque predicates, features are not difficult to locate and change (see post questionnaire analysis, Q5-7 Exp III).
- *Executing the program thwarts Opaque Predicates obfuscation*: Opaque Predicates may be statically undecidable, but at run time their values can be directly observed and the obfuscation can be easily broken. By leveraging this feature, an attacker can understand which segments of the obfuscated code are actually executed and can remove those that are never executed. Obfuscated program execution turned out to be useful for identifier renaming too, because it helped to understand the application behavior anyway (see post questionnaire analysis, Q18).

5.3 General findings

- *Identifier Renaming is preferable to Opaque Predicates*: when it is possible to chose what kind of obfuscation to deploy, Identifier Renaming should be used instead of Opaque Predicates. In fact, by comparing the average time taken to correctly answer a comprehension task on clear and obfuscated code, comprehension tasks on code obfus-

cated with Identifier Renaming require, on average, twice more time than with Opaque predicates (see the analysis of time).
- *Learning is limited on obfuscated code*: Some learning effect was observed. After acquiring some experience in attack tasks, efficiency on tasks improves. However, while improvements are remarkable when attacking clear code, they are very limited on obfuscated code. Obfuscation poses a limit on the amount of knowledge that can be reused between consecutive attack tasks (see co-factor analysis, Lab and Question).

6 Related Work

In the past, the evaluation of the increased complexity introduced by obfuscation has been mainly addressed through code metrics. Collberg *et al.* [9] proposed the use of complexity measures (e.g., *potency*) in obfuscator tools to help developers choose among different obfuscation transformations. More recently, Udupa *et al.* [32] used the amount of time required to perform automatic de-obfuscation to evaluate the usefulness of *control-flow flattening* obfuscation, relying on a combination of static and dynamic analysis. Goto *et al.* [17] proposed the *depth of parse tree* to measures source code complexity. Anckaert *et al.* [1] attempted to quantify and compare the level of protection of different obfuscation techniques. In particular, they proposed 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), however without performing any validation on the proposed metrics. Rather than proposing new metrics, we aim at experimentally assessing obfuscation techniques, by measuring the success of an attack and the efficiency of an attacker in performing it, on both clear and obfuscated source code.

The work most similar to ours is an experimental study on the complexity of reverse engineering binary code [30]. 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 subjects' ability was significantly correlated with the success of reverse engineering tasks they had to perform. Our study goes beyond: we compare—by using statistical tests and effect size measures—the capability and efficiency of subjects in performing attack tasks on clear and obfuscated code. Thus we can quantify the increased effort necessary to reverse engineer an obfuscated program, with respect to the effort necessary for a non-obfuscated one. We also compared two different obfuscation techniques.

In a companion paper [3] we describe the initial design and planning of this experimentation, limited just to one obfuscation technique, i.e., Identifier renaming, and we report early results of Exp I and Exp II, just in terms of attack efficiency. The present work builds on top of it, by extending the experimental design with a new treatment, namely Opaque predicates obfuscation, and three further experiments (Exp III, Exp IV and Exp V). Moreover, we perform analysis of correctness, of time and worst case analysis, and we analyze the effect of co-factors as well as the answers provided by subjects to survey questionnaires over all the five experiments.

7 Conclusions

To the best of our knowledge, this is the first work that presents a family of experiments devoted to quantifying and comparing the effectiveness of code obfuscation, as a countermea-

sure against code tampering. As expected, after enough time, even obfuscated code can be understood and eventually tampered, however the delay due to obfuscation largely depends on which obfuscation technique is used. Quite surprisingly, the simpler obfuscation (Identifier Renaming) was found to be more resilient than the more sophisticated one (Opaque Predicates). This probably depends on the process attackers follow to locate the portions of code to change. These results provide useful hints to design an effective code protection strategy and to integrate code obfuscation with complementary protection approaches, such as code replacement.

Future work will be devoted to replicate this experiment in different contexts. We would like to understand whether (or not) the results obtained by the family of conducted experiments are preserved also for other categories of subjects (e.g., professional developers) and when changing the domain and the complexity of the systems/tasks. Last, but not least, we did not assess the combined effect of different obfuscation techniques: this is another topic of interest for future studies.

References

1. Anckaert, B., Madou, M., Sutter, B.D., Bus, B.D., Bosschere, K.D., Preneel, B.: Program obfuscation: a quantitative approach. In: QoP '07: Proc. of the 2007 ACM Workshop on Quality of protection, pp. 15–20. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1314257.1314263
2. Baker, R.D.: Modern permutation test software. In: E. Edgington (ed.) Randomization Tests. Marcel Decker (1995)
3. Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., Torchiano, M., Tonella, P.: The effectiveness of source code obfuscation: An experimental assessment. In: IEEE 17th International Conference on Program Comprehension (ICPC), pp. 178–187 (2009). DOI 10.1109/ICPC.2009.5090041
4. Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., Torchiano, M., Tonella, P.: The effectiveness of source code obfuscation: an experimental assessment. Tech. rep., University of Sannio – *http://www.rcost.unisannio.it/mdipenta/icpc09-tr.pdf* (2009). URL *http://www.rcost.unisannio.it/mdipenta/icpc09-tr.pdf*
5. Ceccato, M., Preda, M.D., Nagra, J., Collberg, C., Tonella, P.: Barrier slicing for remote software trusting. In: Proc. of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007), pp. 27–36. IEEE Computer Society (Sept. 30 2007-Oct. 1 2007). DOI 10.1109/SCAM.2007.4362895
6. Chang, H., Atallah, M.: Protecting software code by guards. In: ACM Workshop on Security and Privacy in Digital Rights Management. ACM (2002)
7. Cohen, J.: Statistical power analysis for the behavioral sciences (2nd ed.). Lawrence Earlbaum Associates, Hillsdale, NJ (1988)
8. Collberg, C., Nagra, J.: Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection, 1st edn. Addison-Wesley Professional (2009)
9. Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Dept. of Computer Science, The Univ. of Auckland (1997)
10. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 184–196. ACM, New York, NY, USA (1998). DOI http://doi.acm.org/10.1145/268946.268962
11. Cordy, J.: The TXL source transformation language. Science of Computer Programming **61**(3), 190–210 (2006)
12. Devore, J.L.: Probability and Statistics for Engineering and the Sciences. Duxbury Press; 7 edition (2007)
13. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Transactions on Software Engineering **29**(3), 195–209 (2003)
14. Falcarin, P., Collberg, C., Atallah, M., Jakubowski, M.: Guest editors' introduction: Software protection. IEEE Software **28**(2), 24–27 (2011)
15. Falcarin, P., Scandariato, R., Baldi, M.: Remote trust with aspect oriented programming. In: IEEE Advanced Information and Networking Applications (AINA-06). IEEE (2006)
16. Fiutem, R., Tonella, P., Antoniol, G., Merlo, E.: Points-to analysis for program understanding. Journal of Systems and Software **44**(3), 213–227 (1999)

17. Goto, H., Mambo, M., Matsumura, K., Shizuya, H.: An approach to the objective and quantitative evaluation of tamper-resistant software. In: Third Int. Workshop on Information Security (ISW2000), pp. 82–96. Springer (2000)
18. Grissom, R.J., Kim, J.J.: Effect sizes for research: A broad practical approach, 2nd edition edn. Lawrence Earlbaum Associates (2005)
19. Horne, B., Matheson, L., Sheehan, C., Tarjan, R.E.: Dynamic self-checking techniques for improved tamper resistance. In: ACM Workshop on Security and Privacy in Digital Rights Management. ACM (2001)
20. Iversen, G., Norpoth, H.: Analysis of Variance. Sage Publications: second ed. (1987)
21. Juristo, N., Moreno, A.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers, Englewood Cliffs, NJ (2001)
22. Motulsky, H.: Intuitive biostatistics: a nonmathematical guide to statistical thinking. Oxford University Press (2010). URL http://books.google.it/books?id=R477U5bAZs4C
23. Oppenheim, A.N.: Questionnaire Design, Interviewing and Attitude Measurement. Pinter, London (1992)
24. R Core Team: R: A Language and Environment for Statistical Computing. Vienna, Austria (2012). URL http://www.R-project.org. ISBN 3-900051-07-0
25. Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., Ceccato, M.: How developers' experience and ability influence web application comprehension tasks supported by UML stereotypes: A series of four experiments. IEEE Transactions on Software Engineering **36**, 96–118 (2010). DOI http://doi.ieeecomputersociety.org/10.1109/TSE.2009.69
26. Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., Ceccato, M., Visaggio, C.A.: Are fit tables really talking?: a series of experiments to understand whether fit tables are useful during evolution tasks. In: 30th International Conference on Software Engineering (ICSE 2008), pp. 361–370 (2008)
27. Ricca, F., Torchiano, M., Di Penta, M., Ceccato, M., Tonella, P.: Using acceptance tests as a support for clarifying requirements: a series of experiments. Information & Software Technology **51**, 270–283 (2009)
28. Scandariato, R., Ofek, Y., Falcarin, P., Baldi, M.: Application-oriented trust in distributed computing. In: Availability, Reliability and Security, 2008. ARES 08. Third International Conference on, pp. 434–439. IEEE (2008)
29. Sheskin, D.: Handbook of Parametric and Nonparametric Statistical Procedures (4th Ed.). Chapman & All (2007)
30. Sutherland, I., Kalb, G.E., Blyth, A., Mulley, G.: An empirical examination of the reverse engineering process for binary files. Computers & Security **25**(3), 221–228 (2006)
31. Tyma, P.: Method for renaming identifiers of a computer program. US patent 6,102,966 (2000)
32. Udupa, S., Debray, S., Madou, M.: Deobfuscation: reverse engineering obfuscated code. Reverse Engineering, 12th Working Conference on (2005). DOI 10.1109/WCRE.2005.13
33. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering - An Introduction. Kluwer Academic Publishers (2000)