Author(s): Ceccato, Mariano; Di Penta, Massimiliano; Nagra, Jasvir; Falcarin, Paolo; Ricca, Filippo; Torchiano, Marco; Tonell, Paolo.
Article title: The Effectiveness of Source Code Obfuscation: an Experimental Assessment
Year of publication: 2009
Citation: Ceccato, M. et al. (2009) 'The Effectiveness of Source Code Obfuscation: an Experimental Assessment' In: 17th IEEE International Conference on Program Comprehension (ICPC-09), Vancouver (Canada) May 17-19, 2009, IEEE pp 178 - 187
Link to published version: http://dx.doi.org/10.1109/ICPC.2009.5090041
DOI: 10.1109/ICPC.2009.5090041

The Effectiveness of Source Code Obfuscation: an Experimental Assessment

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]University of Sannio, Dept. of Engineering, 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

Source code obfuscation is a protection mechanism widely used to limit the possibility of malicious reverse engineering or attack activities on a software system. Although several code obfuscation techniques and tools are available, little knowledge is available about the capability of obfuscation to reduce attackers' efficiency, and the contexts in which such an efficiency may vary.

This paper reports the outcome of two controlled experiments meant to measure the ability of subjects to understand and modify decompiled, obfuscated Java code, compared to decompiled, clear code.

Results quantify to what extent code obfuscation is able to make attacks more difficult to be performed, and reveal that obfuscation can mitigate the effect of factors that can alter the likelihood of a successful attack, such as the attackers' skill and experience, or the intrinsic characteristics of the system under attack.

Keywords: Empirical studies, Software Obfuscation, Program comprehension

1 Introduction

Software protection has become a central issue for distributed applications providing remote services (e.g., video on demand). In fact, such services are often provided under strict use conditions and service agreement. By tampering with the client code of such applications, attackers may gain personal benefits or unfair treatments and they could access the remote services without complying to the service conditions (e.g., without paying the service fee). Source code obfuscation is widely used to prevent or limit malicious attacks aimed at altering the program's behavior in an illegal or non-permitted way. In fact, attackers can take advantage of static and dynamic analysis methods, first of all to understand the code and locate the features of interest in the implementation, and ultimately to break any protection and implement the modifications necessary to achieve their goals. Attackers can decompile, trace, debug, analyze statically and dynamically and of course inspect the (decompiled) source to perform their task.

Despite the proved theoretical impossibility of building general purpose obfuscators [2], implementations of obfuscators do exist and they are actually used in practice. Available obfuscators provide limited though effective protection against malicious reverse engineering, transforming the code into a semantically equivalent program which is harder to understand for an attacker even if automatic code analysis tools are used [7]. Obfuscation transformations can be classified into three groups [6]: *layout obfuscations*, remove relevant information from the code without changing its behavior; *data obfuscations*, transform application data and data structures (e.g., data encoding, data splitting); and, *control-flow obfuscations*, alter the original flow of the application. The obfuscation technique considered in this work, *identifier renaming*, is an instance of layout obfuscation. It removes relevant information from the code by changing the names of classes, fields and operations into meaningless identifiers.

Apart from common wisdom about the protection capabilities provided by source code obfuscation, very few works dealt with the problem of quantifying the benefits achieved through obfuscation. One of the attributes typically used to measure the strength of obfuscation is the *potency* [6], which represents 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.

We conducted two controlled experiments aimed at empirically assessing the potency of identifier renaming, one of the most widely used obfuscation technique. Specifically, we considered whether the capability to perform comprehension and change tasks was affected by obfuscation. We also quantified the effects of obfuscation, whenever the attack tasks could be successfully completed, by measuring the attacker's efficiency in task execution. In fact, benefits can be obtained from obfuscation even if the attack is not completely prevented, when mounting an attack involves a substantially increased amount of time, compared to breaking the original code. In addition to the average scenario, we take into account the worst-case scenario (i.e., the most effective attack), since in practice the best attacker can easily distribute the instructions to break an application once he is successful. For the same reason, in our experiments we compare the behavior of skilled vs. novice attackers. We also investigate whether there are systems intrinsically easier or harder to attack.

The experiments presented in this paper represent a first contribution toward building a comprehensive, quantitative knowledge on the benefits coming from obfuscation against attackers of various skills and for various kinds of systems being protected. Moreover, other than testing the presence of a significant effect of obfuscation on the attack efficiency, which could easily be expected, this paper deals mainly with determining the *size* of such effect (i.e., the magnitude of the difference between efficiencies experienced with and without obfuscation), and analyzing the interaction of obfuscation with other context factors, such as the attacker's ability and experience, and the characteristics of the system under attack.

The paper is organized as follows: Section 2 gives the details of the experimental design. Experimental results (Section 3) are followed by a discussion, in Section 4, which includes analysis of the threats to validity. After a discussion of related work (Section 5), Section 6 concludes the paper and outlines directions for future work.

2 Experiments 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.* [13].

The *goal* of this study is to analyze the effect of a source code obfuscation technique named *Identifier renaming* with the *purpose* of evaluating its ability in making the code resilient to malicious attacks. The *quality focus* regards how this obfuscation technique reduces 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/he 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 obfuscation technique; and (ii) a practitioner, who wants to ensure high resilience to attacks to subsystems of a distributed application delivered to the clients, running in an untrusted environment.

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, either Master or PhD students. The study consisted in two experiments: Exp I was performed with 10 Master students from the University of Trento and Exp II with 22 PhD students from Politecnico di Torino. Master students 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). All subjects attended at least one software engineering course where they learned analysis, design and testing principles.

The systems 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. 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 (both clear code and obfuscated code). *ChatClient* is a network application that allows people to have text based conversation through the network. Conversations can be public or private. The client consists of 13 classes, for a total of 1030 LOC (both clear and obfuscated code). The obfuscation was performed on the bytecode using the *SandMark* tool[1], which replaces identifiers with randomly generated ones.

2.1 Hypotheses formulation and variable selection

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

H_{01} the source code obfuscation does not significantly decrease the efficiency of an attacker to perform a *comprehension* task.

H_{02} the source code obfuscation does not significantly decrease the efficiency of an attacker to perform a *change* task.

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

Table 1. 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.
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 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.

The above two 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 attacker's efficiency to understand the source code and to perform a change task.

The null hypotheses suggest we have two dependent variables, i.e., the *efficiency in performing comprehension tasks*, and the *efficiency in performing change tasks*. To measure the attacker'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). The above 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 success subjects had in change tasks (*success of change tasks*), we asked them to mount two attacks (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., gaining unlimited fuel, or accessing 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 test suite was defined for each change task. The test suite reproduces the interaction scenario of the attack to be performed.

The efficiency of a subject in performing comprehension or change task is evaluated as:

$$\frac{\sum_{i=1}^{2} Corr_i}{\sum_{i=1}^{2} Time_i} \quad (1)$$

where $Corr_i = 1$ if the $i-th$ comprehension or change task was correctly performed, 0 otherwise, and $Time_i$ is the time in minutes needed to perform the task. In other words, the efficiency is measured as the number of correctly performed task per minute.

The main factor of the experiment—that acts as an independent variable—is the presence of the treatment during the execution of the task. The two alternative treatments are (i) decompiled source code, derived from obfuscated (with identifier renaming) code, and (ii) decompiled clear code. Decompilation was performed using the Jad 1.5 decompiler[2].

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

- *The subjects' Experience:*, i.e., comparing Master and PhD students.
- *The subjects' Ability:* the subjects' Ability was evaluated by resorting to the average grades obtained in the previous related exams. In Italy grades vary between 18 (lowest grade) and 30 (highest grade), however all our subjects had grades above 21. We identified three Ability levels (*low*, *medium*, and *high*) corresponding to the intervals: [22-24], [25-27], and ≥ 28.
- *The System* to be attacked: as detailed in Section 2.2, to use a balanced design we need two systems: Chat-Client and CarRace. Although they 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, we analyze whether there is a learning effect as the subjects perform the four subsequent tasks.

For each co-factor, we test (see Section 2.4) 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, the following two hypotheses are tested:

H_{0c} the co-factor has no significant effect on the efficiency of subjects in performing an attack.

H_{0ci} the co-factor does not significantly interact with the presence of obfuscation to influence the efficiency of subjects in performing the attack.

These hypotheses are two-tailed, because we do not know a-priori the direction of a possible influence of co-factors.

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

Table 2. Experiment design.

	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)

(O) = Obfuscated, (C) = Clear.

2.2 Experiment design

We adopt a balanced experiment design intended to fit two Lab sessions (2-hours each). Subjects are 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 2). Ability levels are equally distributed across groups. 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., analysis of variance) for studying the effect of multiple factors.

2.3 Experimental procedure and material

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 exercises having the purpose of performing comprehension tasks on the (non-obfuscated) source code of an electronic record book. 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.

To perform the experiment, subjects used a personal computer with the Eclipse™ development environment—which they are familiar with—including syntax highlighting and debugger, and the Java API documentation available. We distributed to subjects the following material, available online for replication purposes[3]:

- 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). Subjects had decompiled code available rather than source code because, in a realistic attack, they cannot access source code; instead, they can decompile the binary or the bytecode; 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 experimental package or a longer technical report [4] for details)—most of them expressed in a Likert scale [9]—related to:

- the clarity of tasks and objectives (Q1 – Q4);
- the difficulties experienced when performing the different tasks (comprehension and change) (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 on looking the source code, and on executing the system (Q15, Q16);

[3] http://selab.fbk.eu/ceccato/replication_packages/id_renaming_package.zip

- to what extent subjects considered the analysis of obfuscated code hard (Q17);
- whether subjects considered important executing the system to better understand the behavior of obfuscated code (Q18).

2.4 Analysis method

Different kinds of statistical tests need to be used to analyze the results of this experiment.

Two non-parametric tests are used to test the hypotheses related to differences in the subjects' efficiency in performing comprehension and change tasks (H_{01} and H_{02}). 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 [10]. Given the chosen experiment design, it is also possible to use a paired test, i.e., the Wilcoxon, one-tailed test [10]. 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 allow 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 decreases the attacker's efficiency. To this aim, we used the Cohen d effect size, which 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 [5]. For independent samples—to be used for unpaired analyses—it is defined as the difference between the means (M_1 and M_2), divided by the pooled standard deviation: $d = (M_1 - M_2)/\sigma$. For dependent samples—to be used for paired analyses—it is defined as the difference between the means (M_1 and M_2), divided by the standard deviation of the (paired) differences between samples: (σ_D): $d = (M_1 - M_2)/\sigma_D$.

To provide a picture of what a worst case scenario (best attacker) 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 $\geq \sigma$. 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, i.e., the test of hypotheses H_{0c}, H_{0ci}, as well as the hypothetical effect of confounding factors such as system and lab, is performed using a two-way Analysis of Variance (ANOVA), and interactions are visualized using interaction plots. Although ANOVA is a parametric test, it is considered quite robust also for non-normal and non-interval scale variables.

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). 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 in performing comprehension, feature location, and change tasks (Q5, Q6, Q7), answers of subjects receiving clear code were compared with answers of subjects receiving obfuscated code. In this case a two-tailed Mann-Whitey test is used for the null hypothesis $\widetilde{Q_{Clear}} = \widetilde{Q_{Obfuscated}}$. 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), debuggings (Q14) and time spent looking at the code (Q15) and running the system (Q16).

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

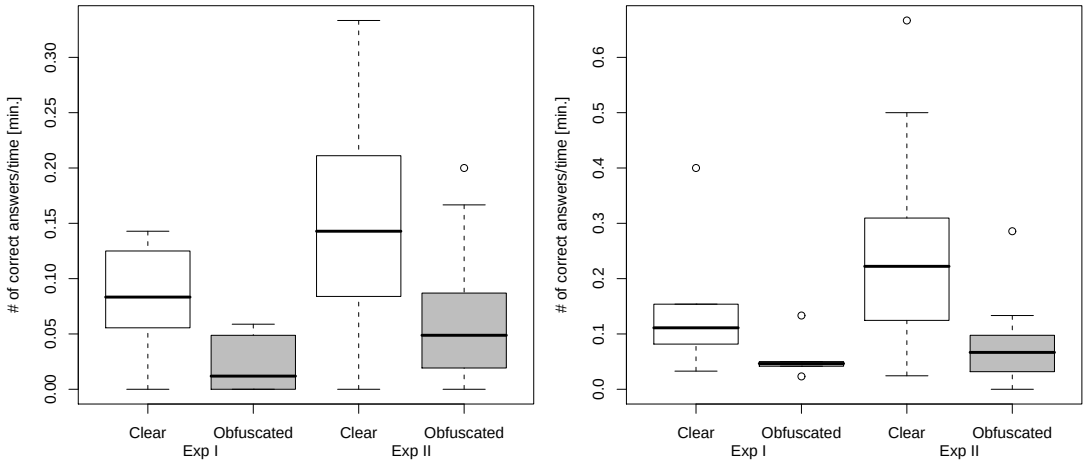
3 Results

This section reports results for the two experiments, with the aim of testing the hypotheses formulated in Section 2.1. Raw data of results are included in the replication package[4], while detailed analyses are reported in a longer technical report [4].

3.1 Efficiency of comprehension and change tasks

Figure 1 shows, for comprehension and change tasks, boxplots of the number of correct answers per minute. Table 3 reports descriptive statistics of the unpaired analysis, including the number of subjects who participated to the experiments, mean, median, standard deviation, Mann-Whitney test p-value and Cohen d effect size. For comprehension tasks, subjects working with clear code significantly outperform subjects with obfuscated code in both Exp I and II, with a *large* effect size (d$\geq$0.8). There are

[4]http://selab.fbk.eu/ceccato/replication_packages/id_renaming_package.zip



(a) Comprehension (b) Change

Figure 1. Boxplots of attack efficiency.

Table 3. Efficiency of attacks.

Exp	Clear				Obfuscated				p	eff.
	N	mean	median	σ	N	mean	median	σ	value	size
I	9	0.08	0.08	0.05	10	0.02	0.01	0.02	$<$**0.01**	1.66
II	19	0.15	0.14	0.10	22	0.06	0.05	0.06	$<$**0.01**	1.02

(a) Comprehension

Exp	Clear				Obfuscated				p	effect
	N	mean	median	σ	N	mean	median	σ	value	size
I	9	0.14	0.11	0.12	10	0.06	0.05	0.04	**0.05**	0.95
II	19	0.24	0.22	0.17	22	0.07	0.07	0.07	$<$**0.01**	1.29

(b) Change

slightly different number of subjects for different treatments because 1 subject in Exp I and 3 subjects in Exp II did not attend the second laboratory. For change tasks, subjects with clear code significantly outperform subjects with obfuscated code in both Exp I (p-value=0.05), and Exp II (p-value $<$0.01). Also for change tasks, the effect size is *high* for both Exp I and II.

In addition to unpaired analysis, we also performed a paired analysis, comparing performances of each subject in the two laboratories, with different treatment and object. This was possible for the 9 subjects of Exp I and 19 subjects of Exp II who attended both labs. Table 4 reports for each experiment the number of subjects, descriptive statistics of differences, Wilcoxon paired test p-value and Cohen d effect size for dependent samples. As shown, again for comprehension tasks there are significant differences in both Exp I and II, with a *large* effect size (d$\geq$0.8) in Exp I and a *medium* effect size (d$\geq$0.5) for Exp II. For change tasks, differences are significant for Exp II only. The effect size is *large* for both experiments. It has to be noticed that 5 subjects out of 9 in Exp I and 2 out of 19 in Exp II were not able to perform the change tasks.

Overall, while unpaired analysis indicates a rejection for both H_{01} and H_{02}, for paired analysis H_{02} cannot be rejected in Exp I.

Table 4. Efficiency: paired analysis.

Exp	N	diff mean	diff median	σ	p value	effect size
I	9	0.06	0.06	0.05	**0.01**	1.07
II	19	0.08	0.08	0.11	$<$ **0.01**	0.78

(a) Comprehension

Exp	N	diff mean	diff median	σ	p value	effect size
I	4	0.14	0.12	0.16	0.12	0.91
II	17	0.19	0.16	0.19	$<$ **0.01**	0.99

(b) Change

3.2 Worst case scenario

Finally, as explained in Section 2, we provide a picture of a worst case scenario by analyzing the lowest times. Results are shown in Table 5, where the pooled standard deviations smaller than the difference of lowest times are highlighted in boldface. We observe two relevant differences for the ChatClient system, both in Exp I. The best time for obfuscated code in T1 is 25 times higher that for clear code, while no one could complete correctly T2 on obfuscated code, compared to 20 minutes required for clear code. As far as CarRace is concerned, we observe relevant differences for T1 in Exp I—15 obfuscated vs. 2 clear—and for T3 in both experiments, 12 vs. 3 in Exp I and 10 vs. 1 in Exp II. In

Table 5. Lowest times for successful attacks.

Exp	Treatment	ChatClient				CarRace			
		T1	T2	T3	T4	T1	T2	T3	T4
I	Clear	1	20	3	15	2	7	3	1
	Obfuscated	25	NA	18	9	15	7	12	3
	σ_{pooled}	**4.8**	**13.7**	23.5	6.2	**3.8**	1.2	**5.0**	5.7
II	Clear	1	3	2	3	2	2	1	1
	Obfuscated	5	2	4	1	4	2	10	<1
	σ_{pooled}	10.8	5.0	8.6	4.9	3.4	5.4	**4.7**	3.6

Times are expressed in minutes.

Table 6. Two-way ANOVA by Treatment & Experience.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	0.09	0.09	17.70	**0.0001**
Experience	1	0.04	0.04	7.21	**0.009**
Treatment:Experience	1	0.00	0.00	0.35	0.56
Residuals	56	0.28	0.00		

(a) Comprehension

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	0.27	0.27	17.19	$<$**0.001**
Experience	1	0.04	0.04	2.27	0.14
Treatment:Experience	1	0.02	0.02	1.02	0.32
Residuals	46	0.72	0.02		

(b) Change

all 5 cases where we observed a relevant difference, higher lowest times were observed for the obfuscated code.

3.3 Analysis of co-factors

This section reports the analysis of co-factors. First, we analyze whether the subjects' Experience (master vs. PhD students) had a significant effect on the efficiency in performing comprehension and change tasks. Table 6 reports results of the two-way ANOVA by Treatment & Experience. For comprehension tasks, the subjects' Experience has a significant effect, although it does not interact with the main factor's treatments. As it can be noted from Figure 1 and from descriptive statistics in Table 3-a, the comprehension level for PhD students (Exp II) is, for both clear and obfuscated code, higher than for Master students (Exp I). For change tasks, the subjects' Experience has no significant effect nor any interaction with the main factor's treatments. However, looking at the interaction plot of Treatment & Experience (for maintenance tasks) in Figure 2, it can be noticed that, when the code is obfuscated, the difference between the less experienced master students and the more experienced PhD students is reduced. If comparing the two experience levels, the effect size is *Medium* (d=0.66) for clear code and *Small* (d=0.26) for obfuscated code.

Similarly, we analyze whether the subjects' Ability (low, medium and high ability levels) had an effect on the efficiency in performing comprehension and change tasks. Results of the two-way ANOVA by Treatment & Ability are

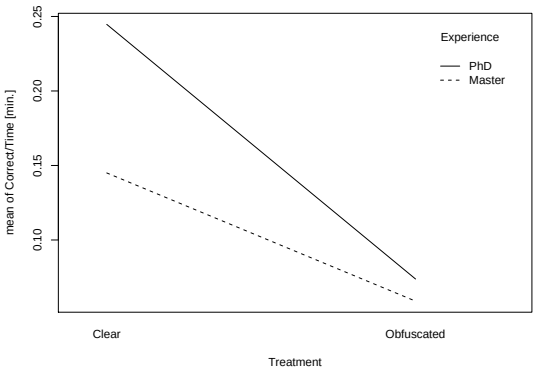


Figure 2. Interaction plot of Treatment & Experience for change tasks.

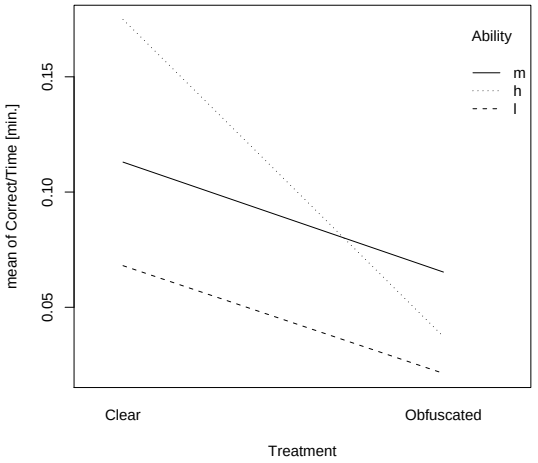


Figure 3. Interaction plot of Treatment & Ability for comprehension tasks.

shown in Table 7. The only effect visible, this time, is an interaction of the Ability with the main factor's treatment for comprehension task, although marginally significant (p-value=0.074). The interaction is clearly visible in the plot of Figure 3: for obfuscated code the gap between low, medium and high Ability subjects is reduced: in other words, the obfuscation reduces the capability of highly skilled subjects to understand the obfuscated code, making them as good as low Ability subjects.

We also analyzed whether the particular System used in the experiment (i.e., CarRace or ChatClient) could have influenced the results. The two-way ANOVA by Treatment & System indicated, for both experiments, a significant influence (p-value=0.01 for Exp I, and 0.001 for Exp II) of the System on the task efficiency, considering both comprehension and change tasks together (results were consistent with those of the two types of task considered separately). If

Table 7. Two-way ANOVA by Treatment & Ability.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	0.09	0.09	17.67	**0.0001**
Ability	2	0.02	0.01	2.00	0.15
Treatment:Ability	2	0.03	0.01	2.73	0.074
Residuals	54	0.27	0.00		

(a) Comprehension

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	0.27	0.27	15.51	**0.0003**
Ability	2	0.01	0.00	0.15	0.86
Treatment:Ability	2	0.00	0.00	0.09	0.91
Residuals	44	0.77	0.02		

(b) Change

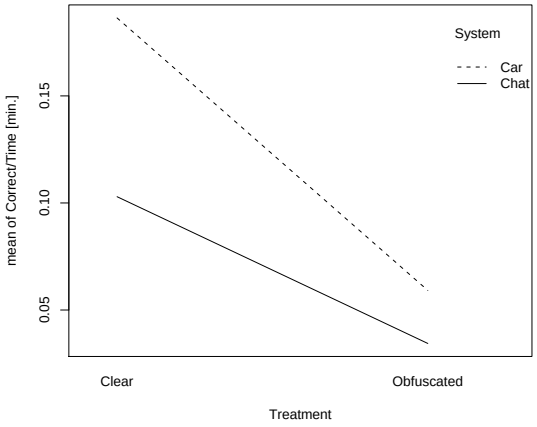


Figure 4. Interaction plot of Treatment & System.

considering together data from both experiments, ANOVA also indicated a mild interaction between System and the main factor's treatment (p-value=0.05). This result can be interpreted by looking at the interaction plot of Figure 4: although the CarRace is always easier to attack than the ChatClient, the difference is reduced when obfuscating the code. In other words, the obfuscation makes systems easier to be attacked (like the CarRace) as difficult as systems that are intrinsically harder to be attacked (like ChatClient).

Finally, we found no significant effect of the Lab factor (p-value=0.40 for Exp I and 0.27 for Exp II) nor any learning across subsequent tasks T1-T4 (p-value=0.36 for Exp I and 0.13 for Exp II).

3.4 Survey questionnaire analysis

Only a few problems emerged from the analysis of survey questionnaire questions (**Q1**, **Q2**, **Q3**, **Q4**, **Q8**, **Q10**) related to the overall subjects' ability to perform the tasks in the time needed and to the clarity of the lab objectives. Subjects of Exp I, regardless of the treatment, experienced problems regarding the time needed to perform the task, and to use the debugger. No particular problem was experienced in Exp II (more experienced subjects).

Then, we compared the answers provided by subjects, when using clear and obfuscated code, 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 the obfuscation 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). In Exp II, subjects felt that obfuscation makes all three activities—comprehension, feature location and change—more difficult (p-value $<$0.01 in all cases). Also, we investigated the perceived usefulness of the main tools available to the subjects, i.e., the use of the debugger (**Q11**) and the renaming facility provided by Eclipse (**Q12**). The debugger was found useful only by subjects in Exp II both for clear and obfuscated code. The renaming facility was found useful by all subjects only for the obfuscated code.

When performing comprehension and change tasks, we investigated whether there was a variation—between subjects with clear and obfuscated code—in the number of system executions (**Q13**) and executions in debugging mode (**Q14**) indicated as needed to perform the task, 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 time when it was obfuscated (**Q13**, p-value=0.04), while in Exp II more executions were performed in debugging mode (**Q14**, p-value=0.01), 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 0.20 in Exp II). Results suggest that in Exp I subjects used system executions as a way to better understand obfuscated systems rather than debugging as subjects of Exp II, since they felt debugging difficult to be performed (as reported in the answers to question **Q10**). Finally, subjects from both experiments agreed (p-value $<$**0.01**) that the obfuscated code was more difficult to understand (**Q17**), and that system execution is necessary for understanding purposes (**Q18**), as a complement to static code analysis.

4 Discussion

Differences in efficiency between subjects receiving clear and obfuscated code are always significant with large effect sizes. If looking at average and median efficiency experienced by subjects with clear and obfuscated code (Table 3), it can be noticed that the efficiency is two to four times higher when having clear code available. This result can be interpreted as follows: when obfuscated code is

available, the attacker would require, to successfully complete an attack, two to four times the time needed with clear code.

Also, results of worst case scenarios indicate that, in most cases, subjects with clear code needed less time than subjects with obfuscated code to successfully complete an attack. Although, as indicated in survey questionnaires, obfuscation makes it more difficult to conduct comprehension, feature location and change tasks, having enough time available, an attacker would be able to achieve his objectives. In most cases the time available to complete an attack is limited, because other protection mechanisms are used in conjunction with obfuscation. For example, (part of) the client code may be periodically replaced/updated with new versions, which makes an attack useless if not performed within the expiration time of a given version.

An important role is played by factors such as the attackers' experience, ability, and the system ease of comprehension and maintenance. Results suggest that:

- The presence of obfuscation reduces the gap between highly skilled or experienced attackers and low skilled/experienced ones. In our experiments, we noticed that the subjects' experience plays a significant role for change tasks (the capability of modifying an unknown, difficult to be maintained system is achieved with years of work), while personal skills influence the ability of performing comprehension tasks. When the code is not obfuscated, highly experienced/skilled subjects perform attacks significantly better than low experienced/skilled ones. The presence of obfuscation makes the system difficult to be attacked for highly skilled/experienced subjects as for low skilled/experienced ones.

- The ability of attackers to understand and change a system is, as one can expect, significantly influenced by the intrinsic characteristics of the system itself. This depends on factors such as the system architecture and design, the presence of adequate requirement and design documentation (not available in both systems used in our experiments) or the quality of the source code (structure, identifiers, comments). What can be noticed when the code is obfuscated is that obfuscation, again, reduces the gap, this time between systems easier to be attacked and systems harder to be attacked.

4.1 Threats to validity

We identified the main threats to the validity that can affect our results: 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. As explained in Section 2, the chosen tasks are representative 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 attacker for that particular attack. Also, the test cases we used only cover 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. Finally, the way subjects' ability was assessed is objective, although we are aware that exam grades may not fully reflect subjects' skills.

Internal validity threats concern external factors that may affect an independent variable. The chosen design allowed us to control a series of factors, such as ability, system, and learning effect. Subjects were not aware of the study hypotheses, and were told not to be evaluated on the performance exhibited during the experiment.

Conclusion validity concerns the relationship between the treatment and the outcome. We used non-parametric tests (Mann-Whitney and Wilcoxon), not requiring data normality. The only parametric test used is the ANOVA which is however robust to deviation from normality. Survey questionnaires, mainly intended to get qualitative insights, were designed using standard ways and scales [9].

External validity concerns the generalization of the findings. First, only one type of obfuscation—identifier renaming—was considered. Work-in-progress aims at experimenting further techniques and performing a comparison among them. 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 study was performed in an academic environment. 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. This threat was at least mitigated (i) by considering graduated students only, (ii) by analyzing the worst case scenario, and (iii) by performing a co-factor analysis by ability. All in all, many hackers are not that different from best students (high Ability subjects/worst case scenarios in our experiments).

5 Related Work

In the past, the evaluation of the increased complexity introduced by obfuscation has been mainly addressed through code metrics. Collberg *et al.* [6] proposed the use of complexity measures (e.g., *potency*) in obfuscator tools to help developers choosing among different obfuscation transformations. More recently, Udupa *et al.* [12] 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.* [8] proposed the *depth of parse tree* to measures source code complexity. Anckaert *et al.* [1] attempted at quantifying and comparing 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 more similar to ours is an experimental study on the complexity of reverse engineering binary code [11]. 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: first, 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.

In a companion paper [3] we describe the design and planning of this experimentation, and briefly summarize early results (only unpaired analysis) of Exp I. The present work extends it by adding a further experiment (Exp II), reporting paired analysis, and analyzing the effect of cofactors and the answers provided by subjects to survey questionnaires.

6 Conclusions

We have described and discussed the results obtained from two experiments, designed to evaluate the potency of source code obfuscation. In particular, we focused on identifier renaming, one of the most widely used obfuscation methods.

Results show that identifier renaming significantly decreases the efficiency of attacks, at least doubling the time needed to complete a successful attack (even in the worst-case scenario, i.e., against the best attacker). In addition, obfuscation reduces the gap between novice and skilled attackers, making the latter less efficient, and makes systems that are easier to attack in clear more similar to those that are intrinsically harder to break.

Our future work will include replication of the study in different settings and contexts, so as to corroborate our findings and extend their validity. In particular, we plan to assess other obfuscation techniques (e.g., opaque predicates), by performing a comparative assessment of their effects.

References

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

[2] 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] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. Towards experimental evaluation of code obfuscation techniques. In *Proc. of the 4th Workshop on Quality of Protection*, pages 39–46. ACM, Oct 2008.

[4] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. The effectiveness of source code obfuscation: an experimental assessment. Technical report, University of Sannio – *http://www.rcost.unisannio.it/mdipenta/icpc09-tr.pdf*, Jan 2009.

[5] J. Cohen. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

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

[7] C. Collberg, C. Thomborson, and D. Low. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, 28, 2002.

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

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

[10] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures (4th Ed.)*. Chapman & All, 2007.

[11] 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.

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

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