Programming Experience Might Not Help in Comprehending Obfuscated Source Code Efficiently

Norman Hänsch\textsuperscript{1}, Andrea Schankin\textsuperscript{2}, Mykolai Protsenko\textsuperscript{3}
Felix Freiling\textsuperscript{1}, Zinaida Benenson\textsuperscript{1}

\textsuperscript{1}Friedrich-Alexander-Universität Erlangen-Nürnberg
\textsuperscript{2}Karlsruhe Institute of Technology
\textsuperscript{3}Fraunhofer Institute for Applied and Integrated Security

August 14, 2018
Software obfuscation: protect programs by making them harder to understand
Background - Software Obfuscation

- Software obfuscation: protect programs by making them harder to understand.
  Who obfuscates? Why?

- Collberg et al. (Technical Report, 1997):
  - Potency: To what degree is a human reader confused?
  - Resilience: How well does an obfuscation method withstand an attack from an automatic deobfuscator?
  - Evaluation based on software metrics

- Ceccato et al. (Empirical Software Engineering, 2014): user studies
Background - Software Obfuscation

- Software obfuscation: protect programs by making them harder to understand

  **Who obfuscates?**  **Why?**
  
  Software vendors  secure their intellectual property
  Hackers  make their malicious code harder to understand
Background - Software Obfuscation

- Software obfuscation: protect programs by making them harder to understand
  
  **Who obfuscates?**  
  **Why?**
  
  Software vendors  
  secure their intellectual property
  
  Hackers  
  make their malicious code harder to understand

- Collberg et al. (Technical Report, 1997):
  
  - Potency: To what degree is a human reader confused?
  - Resilience: How well an obfuscation method stands up under an attack from an automatic deobfuscator?
Background - Software Obfuscation

- Software obfuscation: protect programs by making them harder to understand

<table>
<thead>
<tr>
<th>Who obfuscates?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software vendors</td>
<td>secure their intellectual property</td>
</tr>
<tr>
<td>Hackers</td>
<td>make their malicious code harder to understand</td>
</tr>
</tbody>
</table>

- Collberg et al. (Technical Report, 1997):
  - Potency: To what degree is a human reader confused?
  - Resilience: How well an obfuscation method up under an attack from an automatic deobfuscator?
  - Evaluation based on software metrics
Background - Software Obfuscation

- Software obfuscation: protect programs by making them harder to understand

<table>
<thead>
<tr>
<th>Who obfuscates?</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software vendors</td>
<td>secure their intellectual property</td>
</tr>
<tr>
<td>Hackers</td>
<td>make their malicious code harder to understand</td>
</tr>
</tbody>
</table>

- Collberg et al. (Technical Report, 1997):
  - Potency: To what degree is a human reader confused?
  - Resilience: How well an obfuscation methods up under an attack from an automatic deobfuscator?
  - Evaluation based on software metrics

- Ceccato et al. (Empirical Software Engineering, 2014): user studies
The Study - Replication and Novelty

Replication Study materials
Questionnaire
Code understanding (correctness, time, efficiency)
Obfuscation methods: Name Overloading (NO), Opaque Predicates (OP)
The Study - Replication and Novelty

Replication
- Study materials
- Questionnaire
- Code understanding (correctness, time, efficiency)
- Obfuscation methods: Name Overloading (NO), Opaque Predicates (OP)

Novelty
- Code analysis behavior (actions & time spent on them)
- The influence of experience
- Evaluation of correctness of the answers
- Study design
The Study - The Programs

Figure: Race Program

Figure: Chat Program
Listing 1: Code obfuscated with Name Overloading (NO)

```java
public void __m1(int i)
{
  if(__f22)
    if(__f19 == 0)
      {
        __f5 += i;
        if(__f5 > __f6 / 10)
          __f5 = __f6 / 10;
        else
          if(__f5 < __f7 / 10)
            __f5 = __f7 / 10;
      }
  else
    [...]
```

Listing 2: Clear code from Race MovingCarModel.java

```java
public void changeSpeed(int i) {
    if (started)
        if (gas == 0)
            {
                speed += i;
                if (speed > maxSpeed / 10)
                    speed = maxSpeed / 10;
                else
                    if (speed < minSpeed / 10)
                        speed = minSpeed / 10;
            } else [...]
```
The Study - Code Examples

Listing 3: Code obfuscated with Opaque Predicates (OP)

```java
public void changeSpeed(int i) {
    if (Node.getI() != Node.getH()) {
        lastFuel = (0L + time2) - (long) lap;
        started = lastFuel == 0L;
        Node.getF().setLeft(Node.getH().getLeft());
    } else {
        Node.getG().getLeft().swap(
            Node.getG().getRight());
        if (started)
            if (Node.getI() == Node.getH()) {
                if (gas == 0) {
                    if (Node.getF() == Node.getG()) {
                        Node.getF().setLeft(
                            Node.getI().getRight());
                        [...]
                }
```
The Study - Study Design

<table>
<thead>
<tr>
<th>Group</th>
<th>1st Program (clear code)</th>
<th>2nd Program (obfuscated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Race: $Rnd$(Box,Laps)</td>
<td>NO(Chat): $Rnd$(Messages,Users)</td>
</tr>
<tr>
<td>2</td>
<td>Race: $Rnd$(Box,Laps)</td>
<td>OP(Chat): $Rnd$(Messages,Users)</td>
</tr>
<tr>
<td>3</td>
<td>Chat: $Rnd$(Messages,Users)</td>
<td>NO(Race): $Rnd$(Box,Laps)</td>
</tr>
<tr>
<td>4</td>
<td>Chat: $Rnd$(Messages,Users)</td>
<td>OP(Race): $Rnd$(Box,Laps)</td>
</tr>
</tbody>
</table>
The Study - Demographics of the Participants

- 66 participants
  - 44 bachelor students
  - 20 master students
  - 2 PhD students
- 24.2% already participated in a course related to software obfuscation
Results - Code Comprehension

<table>
<thead>
<tr>
<th></th>
<th>Clear vs NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>-0.113</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-0.312*</td>
</tr>
<tr>
<td>Total time</td>
<td>0.338**</td>
</tr>
<tr>
<td>Time correct</td>
<td>0.351*</td>
</tr>
</tbody>
</table>

**Table:** Wilcoxon & Mann-Whitney-U tests; *$p < .05$, **$p < .01$**

Effect sizes $r$ (no effect, small, medium)
## Results - Code Comprehension

### Table: Wilcoxon & Mann-Whitney-U tests; *$p < .05$, **$p < .01$
Effect sizes $r$ (no effect, small, medium)

<table>
<thead>
<tr>
<th></th>
<th>Clear vs NO</th>
<th>Clear vs OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>-0.113</td>
<td>-0.154</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-0.312*</td>
<td>-0.332**</td>
</tr>
<tr>
<td>Total time</td>
<td>0.338**</td>
<td>0.276*</td>
</tr>
<tr>
<td>Time correct</td>
<td>0.351*</td>
<td>0.193</td>
</tr>
</tbody>
</table>
Results - Code Comprehension

<table>
<thead>
<tr>
<th></th>
<th>Clear vs NO</th>
<th>Clear vs OP</th>
<th>NO vs OP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>-0.113</td>
<td>-0.154</td>
<td>-0.079</td>
</tr>
<tr>
<td>Efficiency</td>
<td>-0.312*</td>
<td>-0.332**</td>
<td>0.045</td>
</tr>
<tr>
<td>Total time</td>
<td>0.338**</td>
<td>0.276*</td>
<td>-0.156</td>
</tr>
<tr>
<td>Time correct</td>
<td>0.351*</td>
<td>0.193</td>
<td>-0.260</td>
</tr>
</tbody>
</table>

**Table:** Wilcoxon & Mann-Whitney-U tests; *p < .05, **p < .01

Effect sizes r (no effect, small, medium)
Results - Behavior

Clear vs NO

<table>
<thead>
<tr>
<th>Number of:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>File open commands</td>
<td>0.451**</td>
</tr>
<tr>
<td>Advanced commands</td>
<td>0.352**</td>
</tr>
<tr>
<td>Program executions</td>
<td>0.243</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.420**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time spent on:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Program executions</td>
<td>0.290*</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.433**</td>
</tr>
<tr>
<td>Code reading</td>
<td>0.080</td>
</tr>
</tbody>
</table>

Table: Wilcoxon & Mann-Whitney-U tests; *p < .05, **p < .01
Effect sizes r (no effect, small, medium)
# Results - Behavior

<table>
<thead>
<tr>
<th></th>
<th>Clear vs NO</th>
<th>Clear vs OP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File open commands</td>
<td>0.451**</td>
<td>0.058</td>
</tr>
<tr>
<td>Advanced commands</td>
<td>0.352**</td>
<td>0.282*</td>
</tr>
<tr>
<td>Program executions</td>
<td>0.243</td>
<td>0.356**</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.420**</td>
<td>0.349**</td>
</tr>
<tr>
<td><strong>Time spent on:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program executions</td>
<td>0.290*</td>
<td>0.278*</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.433**</td>
<td>0.308*</td>
</tr>
<tr>
<td>Code reading</td>
<td>0.080</td>
<td>0.016</td>
</tr>
</tbody>
</table>

*Table:* Wilcoxon & Mann-Whitney-U tests; *p < .05, **p < .01

Effect sizes r (no effect, small, medium)
## Results - Behavior

<table>
<thead>
<tr>
<th></th>
<th>Clear vs NO</th>
<th>Clear vs OP</th>
<th>NO vs OP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File open commands</td>
<td>0.451**</td>
<td>0.058</td>
<td>-0.373**</td>
</tr>
<tr>
<td>Advanced commands</td>
<td>0.352**</td>
<td>0.282*</td>
<td>-0.106</td>
</tr>
<tr>
<td>Program executions</td>
<td>0.243</td>
<td>0.356**</td>
<td>-0.104</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.420**</td>
<td>0.349**</td>
<td>0.030</td>
</tr>
<tr>
<td><strong>Time spent on:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program executions</td>
<td>0.290*</td>
<td>0.278*</td>
<td>-0.014</td>
</tr>
<tr>
<td>Debugging mode</td>
<td>0.433**</td>
<td>0.308*</td>
<td>-0.079</td>
</tr>
<tr>
<td>Code reading</td>
<td>0.080</td>
<td>0.016</td>
<td>-0.081</td>
</tr>
</tbody>
</table>

Table: Wilcoxon & Mann-Whitney-U tests; *p < .05, **p < .01
Effect sizes r (no effect, small, medium)
The Study - Experience

- **Survey questions**
  - *Programming Experience*: quality and type of code written so far
  - *Obfuscation Experience*: experience with obfuscation and debugging
  - *Java Experience*: experience with Java and using Eclipse

- **Experiment**
  - *Comprehension Skills*: efficiency in working on clear code
The Study - Experience

- **Survey questions**
  - *Programming Experience*: quality and type of code written so far
  - *Obfuscation Experience*: experience with obfuscation and debugging
  - *Java Experience*: experience with Java and using Eclipse

- **Experiment**
  - *Comprehension Skills*: efficiency in working on clear code

- **k-means cluster analysis:**
  - 21 beginners
  - 45 experienced participants
Experience leads to significant differences concerning:

<table>
<thead>
<tr>
<th>$\omega^2$</th>
<th>$p$</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16</td>
<td>**</td>
<td>Correctness</td>
</tr>
<tr>
<td>0.10</td>
<td>**</td>
<td>Efficiency</td>
</tr>
<tr>
<td>0.13</td>
<td>**</td>
<td>Advanced commands</td>
</tr>
<tr>
<td>0.13</td>
<td>**</td>
<td>Debugging mode</td>
</tr>
<tr>
<td>0.05</td>
<td>*</td>
<td>Time spent debugging</td>
</tr>
</tbody>
</table>

Table: ANOVA; *$p < .05$, **$p < .01$; Effect size $\omega^2$ (small, medium, large effect)
Results - Exp. x Obf.: Behavior

Figure: $\omega^2 = 0.06^*$ significant difference (ANOVA; $^*p < .05$, $^{**}p < .01$.)
Results - Exp. x Obf.: Behavior

Figure: $\omega^2 = 0.09^{**}$ significant difference (ANOVA; *$p < .05$, **$p < .01$.)
Results - Exp. x Obf.: Code Comprehension

Figure: $\omega^2 = 0.01^*$ significant difference (ANOVA; $^*p < .05$, $^{**}p < .01.$)
Summary

1. Confirmation of all findings by Ceccato et al.
2. Empirical support of the taxonomy of Collberg et al.
3. Code comprehension behavior on obfuscated software may be different from comprehension on traditional programs.
4. Programming experience might not help in comprehending obfuscated source code efficiently.