How does code obfuscation impact energy usage?

Cagri Sahin1, Mian Wan2, Philip Tornquist1, Ryan McKenna1, Zachary Pearson1, William G. J. Halfond2 and James Clause1,*

1Computer Science Department, University of Delaware, Newark, DE, USA
2Department of Computer Science, University of Southern California, Los Angeles, CA, USA

ABSTRACT

Software piracy is an important concern for application developers. Such concerns are especially relevant in mobile application development, where piracy rates can be greater than 90%. The most common approach used by mobile developers to prevent piracy is code obfuscation. However, the decision to apply such transformations is currently made without regard to the impacts of obfuscations on another area of increasing concern for mobile application developers, energy usage. Because both software piracy and battery life are important concerns, mobile application developers must strike a balance between protecting their applications and preserving the battery lives of their users’ devices. To help them make such choices, we conducted an empirical study of the effects of 18 code obfuscations on the amount of energy consumed by executing a total of 21 usage scenarios spread across 11 Android applications on four different mobile phone platforms. The results of the study indicate that, while obfuscations can have a statistically significant impact on energy usage and are more likely to increase energy usage than to decrease energy usage, the magnitudes of such impacts are unlikely to be meaningful to mobile application users. Copyright © 2016 John Wiley & Sons, Ltd.

1. INTRODUCTION

Software piracy is an important concern for application developers. A 2011 study conducted by the Business Software Alliance found that over 40% of software is pirated, resulting in a potential loss of over $63bn (http://globalstudy.bsa.org/2011/index.html). Such concerns are especially relevant in mobile application development, where piracy rates can top 90% (http://www.businessinsider.com/android-piracy-problem-2015-1). The most common approach used by mobile developers to prevent piracy is code obfuscation—making the code of their applications more difficult for a human to understand. Both Microsoft and Google strongly recommend that developers obfuscate their applications (http://developer.android.com/tools/help/proguard.html; http://www.microsoft.com/en-us/download/details.aspx?id=7490). Google has even gone so far as to integrate obfuscation into the standard Android build system.

In practice, there are many different types of transformations (e.g., renaming variables and methods; merging, splitting, and reordering code; and complicating control and data flow) that are used to successfully obfuscate code. However, the decision to apply such transformations is performed without regard to their impacts on another area of increasing concern for mobile application...
developers, energy usage. As a result, an obfuscated application may consume an excessive amount of power, draining the battery and causing users to leave poor reviews or to request refunds (http://apigee.com/about/pressrelease/apigee-survey-users-reveal-top-frustrations-lead-bad-mobile-app-reviews).

Because both software piracy and battery life are important concerns, mobile application developers must strike a balance between (i) protecting their applications and intellectual property and (ii) preserving the limited battery power of the devices where their applications will execute. A major obstacle to striking an appropriate balance between these concerns is a lack of information about how changes to an application impact its energy usage. As a result, developers must either make a poorly informed choice or, more commonly, use an obfuscation tool’s default configuration. Unfortunately, these approaches often result in applications that either consume more energy than necessary or are not protected as effectively as they could be.

To address the lack of information available to developers, we conducted an initial investigation into the energy impacts of applying different code obfuscations [1]. This paper presents an empirical study that improves on this prior work in several ways. First, we extended the original study by adding six additional user scenarios, increasing the number of considered scenarios from 15 to 21. Second, we repeated the extended study on three additional mobile phone platforms, increasing the number of considered platforms from one to four. Appropriate text and figures describing the extensions were added to Sections 2 and 3. Together, these extensions improve the generalizability of the study by allowing us to investigate the impacts of obfuscations in a wider range of use cases and platforms. In addition, to extending and repeating the original study, we also provide a more detailed description of our energy measurement platforms (EMPs) and expanded and updated Section 4.

At a high level, the study presented in this paper is focused on addressing two major questions: (i) How do different obfuscations alter the overall energy usage of an application? and (ii) Are the impacts of obfuscations likely to be meaningful for mobile application users? To answer these questions, we used four obfuscation tools to create a total of 198 obfuscated versions of 11 Android applications. We executed each obfuscated version multiple times on four different mobile phone platforms with one or more different user scenarios and measured the amount of energy that was consumed. In total, we ran over 47,000 executions to gather the experimental data. We then performed a statistical analysis of the collected data to investigate the impacts of the obfuscations on energy usage and answer our research questions.

The results of our study demonstrate the following:

1. Obfuscations can, and often do, impact the energy usage of applications.
2. Obfuscations can both increase and decrease energy usage, but they are much more likely to increase energy usage.
3. The magnitude of the impacts of obfuscations is comparable with the magnitude of the impacts of other code level changes, such as applying refactorings.
4. The differences between the impacts of the considered obfuscation configurations and obfuscation tools on energy usage are not statistically significant.
5. The impacts of obfuscation on battery life are unlikely to be meaningful to mobile application users.

These results are positive for both mobile application users and mobile application developers. Developers can protect their applications without impacting the battery life of the devices where their applications execute.

The remainder of this paper is organized as follows: Section 2 describes the methodology of our study including our independent and dependent variables, considered applications and scenarios, obfuscation approaches, and data collection protocol. Section 3 presents and discusses the results of the study including potential threats to its validity. Finally, Sections 4 and 5 discuss related work and present our conclusions and future work.

2. EMPIRICAL STUDY

This section describes the details of our study design, including our independent and dependent variables, considered applications and scenarios, obfuscation approaches, and data collection...
protocol. In planning this work, we followed well-known guidelines for empirical study design [2]. All of our experimental applications, artifacts, and summary data are publicly available at https://bitbucket.org/udse/obfuscation-study. Our raw experimental data are too large to host publicly but are available upon request.

2.1. Experimental variables

In this study, we considered one dependent variable (the amount of energy consumed by the execution of an application) and one independent variable (the obfuscation applied to the application). To isolate the impacts of changing our independent variable on our dependent variable, it was necessary to control for the effects of several extraneous variables (e.g., unnecessary changes in the considered application’s code and the inputs used to drive the application). The remainder of this section describes how we controlled for such extraneous variables.

2.1.1. Controlling for extraneous changes in an application’s code. In many cases, obfuscations are not formally specified. Because of this, different tools may use the same name to refer to different sequences of code changes. For example, many obfuscation tools provide a transformation called string encryption. At a high level, all of these transformations perform the same operation: encrypting the constant strings in an application so that they cannot be easily understood. However, the specific encryption algorithm used can vary greatly. This flexibility in nomenclature can be a potential source of bias and a potential source of confusion in interpreting the results of the study. If we compared the impacts of obfuscations that were inconsistently applied, we would essentially be comparing different transformations. Similarly, if a developer would apply a substantially different set of code edits that happen to share the same name as one of the obfuscations that we considered, the results that they observe could be drastically different than what we observed.

In order to avoid these potential problems, we ensured that all obfuscations were applied in a consistent, repeatable, and well-documented manner. To accomplish this, we relied on several commonly used obfuscation tools (Section 4). By using preexisting automated tools, we are ensuring that the changes we are making to our considered applications are the same changes that a developer would apply if they applied the same obfuscations using the same tool.

2.1.2. Controlling for inconsistencies in executing an application. In general, mobile applications are interactive and event driven. They accept input, either from a user or from a sensor, perform some computation, and generate a response. In our experiments, this interactive nature can introduce a potential source of bias as it is difficult to manually reproduce a given execution exactly. For example, a user can often repeatedly perform the same sequence of actions (e.g., enter text into a textbox or click a button) but cannot maintain the same timing between the actions. Although such differences may seem inconsequential, they may lead to observed differences in energy usage that are not due to changing our independent variable, but rather to differences in how the application is driven. In order to prevent such bias, it is necessary to be able to reproduce deterministically a given sequence of actions with great fidelity. Capture/replay tools provide this functionality.

Capture/replay tools are designed to allow for the deterministic replay of a sequence of recorded events. Conceptually, this is accomplished by wrapping an application to insulate it from its environment. When capturing, the wrapper records all of the events that are passed to the application from the environment. When replaying, the wrapper replaces the environment and passes the recorded events to the application. Because precise timing information is recorded during the capture process, there is very little variability in when events are passed to the application during replay. Hence, when using a capture replay tool, any observed variations in energy usage are more likely to be the result of the obfuscations used rather than inconsistencies in driving the application.

We chose to use RERAN [3] as our capture/replay tool, because it is designed to record and replay Android applications. Also, RERAN has a lightweight implementation, and its run-time overhead is low, close to 1%.
2.2. Considered applications

As the applications for our study, we used popular, easily accessible Android applications. We selected Android applications for several reasons. First, Android application developers typically care about both the security of their intellectual property and the energy efficiency of their applications. Second, there are many existing obfuscation tools that specifically target Android applications, or, more generally, operate on Java code, that we can use. Third, the source code of many Android applications is freely available, allowing us to easily create many different obfuscated versions. Finally, we have extensive infrastructure to run Android applications and measure their energy usage.

Table I lists the specific applications that we selected. The first two columns, Application and Description, list the name of each application and a brief description of its functionality. The third column, LoC, shows the application’s number of lines of code; and the final column, Size, shows the size of the application’s compiled application package file. The LoC measurement includes only the application itself, while the size measurement includes both the application and its necessary libraries. Because our considered obfuscation tools obfuscate both the application and its libraries, even when the source of such libraries is unavailable, we chose to report both measures to give a better understanding of the amount of code that is being obfuscated.

We choose these specific applications for several reasons. First, they are representative of a wide variety of common application types (e.g., games, study aids, and productivity tools). Second, they are popular and widely used. For example, calculator, calendar, and clock are part of the default Android installation. Finally, they are supported by RERAN. Although RERAN is generally effective at replaying user inputs, such as touch events, it does not support replaying network connections or other sensor readings (e.g., GPS). As such, we were unable to include applications that depend on these types of inputs.

Note that in order to experiment with these applications successfully, we needed to modify them slightly. Primarily, the modifications were made to their build systems so that we could automate the obfuscation processes, but in some cases we also needed to modify the application’s source code to remove sources of randomness that are not handled by RERAN (e.g., we modified the random number generator to use a fixed seed).

2.3. Considered usage scenarios

Our considered applications are driven primarily by user input. To create the inputs necessary for driving the applications, we examined each application and created one or more usage scenarios. Our goal in creating these scenarios was to capture what we believe to be typical usage patterns for the application (i.e., actions that users are likely to perform). By focusing on typical scenarios rather than scenarios designed to maximize other metrics such as coverage, we were able to gain a better understanding of the impacts of obfuscations on a user’s daily interactions with their mobile device.

Table II shows the specific usage scenarios that we created. The first two columns, Application and Name, show the application that is used in the scenario and a distinguishing name. For example, AnkiDroid has two scenarios, AnkiDroid: New Deck and AnkiDroid: Tutorial Deck. The third

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>LoC</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnkiDroid</td>
<td>Flashcard application</td>
<td>44,913</td>
<td>2.4</td>
</tr>
<tr>
<td>Calculator</td>
<td>Default Android calculator</td>
<td>1427</td>
<td>2.6</td>
</tr>
<tr>
<td>Calendar</td>
<td>Default Android calendar</td>
<td>41,715</td>
<td>1.4</td>
</tr>
<tr>
<td>Clock</td>
<td>Default Android clock</td>
<td>13,477</td>
<td>1.0</td>
</tr>
<tr>
<td>DailyMoney</td>
<td>Daily financial tracker</td>
<td>8723</td>
<td>0.4</td>
</tr>
<tr>
<td>FrozenBubblePlus</td>
<td>Bubble popping puzzle game</td>
<td>7517</td>
<td>0.2</td>
</tr>
<tr>
<td>Nim</td>
<td>Mathematical strategy game</td>
<td>1475</td>
<td>0.8</td>
</tr>
<tr>
<td>OIFileManager</td>
<td>File manager</td>
<td>7200</td>
<td>0.7</td>
</tr>
<tr>
<td>OpenSudoku</td>
<td>Sudoku game</td>
<td>6079</td>
<td>0.2</td>
</tr>
<tr>
<td>SkyMap</td>
<td>Astronomy application</td>
<td>10,921</td>
<td>0.7</td>
</tr>
<tr>
<td>Tomdroid</td>
<td>Note-taking application</td>
<td>7955</td>
<td>0.6</td>
</tr>
</tbody>
</table>
column, Description, provides a brief description of what user actions are performed during the scenario. For example, during the AnkiDroid: New Deck scenario, a new flash card deck is created, and five flash cards are added to the newly created deck. In total, we created 21 scenarios for our applications: three for clock, OIFileManager, and SkyMap; two for AnkiDroid, calculator, Dailymoney, and OpenSudoku; and one for calendar, FrozenBubblePlus, Nim, and Tomdroid.

2.4. Considered obfuscations

2.4.1. Obfuscation tools. We had two requirements when choosing obfuscation tools. These were the tools that could (i) obfuscate Android applications and (ii) be easily integrated into the standard Android build system. Because we are repeatedly obfuscating multiple applications, manually applying obfuscations is infeasible. Unfortunately, these requirements eliminated the majority of the free or open source Java obfuscation tools. While such tools can work well for standard Java software, either they introduce changes that result in invalid Android applications when the obfuscated class files are converted to the dex format or they cannot be integrated into the Android build system. The only free obfuscation tool that we found that met our requirements was Proguard 4.10 (http://proguard.sourceforge.net), which is the obfuscation tool that is bundled with the Android Software Development Kit.

Because of the limited number of free tools that met our requirements, we also considered commercial obfuscation tools. Here, we found tools that were more likely to fulfill our requirements. However, their trial or evaluation versions are often limited in functionality (e.g., they only obfuscate parts of an application, or do not support the full suite of configuration options). As such, they are not suitable for our study. To obtain full-featured versions, we emailed the tool developers and asked if they would be willing to donate a copy of their obfuscation tool. As the result of this process, we obtained copies of three commercial obfuscation tools: Allatori 4.7 (http://www.allatori.com), DashO 7.2 (http://www.preemptive.com/products/dasho), and Zelix KlassMaster 6.1.3 (ZKM) (http://www.zelix.com/klassmaster/).

2.4.2. Obfuscation configurations. After reading the manuals of Allatori, DashO, Proguard, and ZKM, we identified several common high-level configurations or obfuscation types:

<table>
<thead>
<tr>
<th>Application</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnkiDroid</td>
<td>New deck</td>
<td>Create a new slide deck containing 5 cards.</td>
</tr>
<tr>
<td></td>
<td>Tutorial deck</td>
<td>Review the 20 cards in the tutorial deck.</td>
</tr>
<tr>
<td>Calculator</td>
<td>Advance</td>
<td>Perform several advanced arithmetic calculations.</td>
</tr>
<tr>
<td></td>
<td>Standard</td>
<td>Perform several basic arithmetic calculations.</td>
</tr>
<tr>
<td>Calendar</td>
<td>Add event</td>
<td>Add a new event, search for it, delete it.</td>
</tr>
<tr>
<td></td>
<td>Interval</td>
<td>Create intervals while running the stopwatch.</td>
</tr>
<tr>
<td></td>
<td>Stopwatch</td>
<td>Run the stopwatch for 10 s.</td>
</tr>
<tr>
<td></td>
<td>Timer</td>
<td>Run a 10-s countdown timer.</td>
</tr>
<tr>
<td>DailyMoney</td>
<td>Add detail</td>
<td>Enter two transactions.</td>
</tr>
<tr>
<td></td>
<td>View lists</td>
<td>View details and balances.</td>
</tr>
<tr>
<td>FrozenBubblePlus</td>
<td>Level 1</td>
<td>Play the first level.</td>
</tr>
<tr>
<td>Nim</td>
<td>Easy AI</td>
<td>Play 3 rounds with increasing difficulty levels.</td>
</tr>
<tr>
<td>OIFileManager</td>
<td>Create file</td>
<td>Create 2 folders, nest folders, delete folders.</td>
</tr>
<tr>
<td></td>
<td>Play file</td>
<td>View 4 pictures and play a ringtone 3 times.</td>
</tr>
<tr>
<td></td>
<td>View file</td>
<td>Open a file. Navigate directories.</td>
</tr>
<tr>
<td>OpenSudoku</td>
<td>Easy level 1</td>
<td>Complete a single easy Sudoku grid.</td>
</tr>
<tr>
<td></td>
<td>Hard level 1</td>
<td>Complete a single hard Sudoku grid.</td>
</tr>
<tr>
<td>SkyMap</td>
<td>Find Mars</td>
<td>Set time to a fixed past date, searches for Mars.</td>
</tr>
<tr>
<td></td>
<td>Move zoom</td>
<td>Arbitrarily zoom in/out, moves along the map.</td>
</tr>
<tr>
<td></td>
<td>Show component</td>
<td>Show each component, toggle night mode.</td>
</tr>
<tr>
<td>Tomdroid</td>
<td>Notes</td>
<td>Create a note, search for text, open the note, delete the note.</td>
</tr>
</tbody>
</table>
• Control-flow (cf): Produces spaghetti logic that is difficult or impossible to decompile by inserting branching and conditional instructions into the body of a method.
• Rename (rename): Renames packages, classes, methods, and fields to short meaningless names (e.g., a and b) and, if possible, moves classes into a single package.
• Optimize (opt): Removes unused classes, fields, methods, and attributes; performs simple bytecode optimizations (e.g., peephole optimizations); and removes dead code.
• String encryption (se): All constant strings in the application are replaced with an encrypted version; decryption methods are added so that strings can be decrypted at runtime.
• All (all): Combines all other configurations supported by an obfuscation tool.

Information about the effectiveness of these types of obfuscations can be found in related work (e.g., [4, 5]). Note that, while the specific changes made by each tool for each configuration may vary (e.g., different string encryption algorithms may be used or branches may be inserted in different locations), from the point of view of an application developer, the results are essentially identical. In addition, not every configuration is supported by every tool.

Table III shows which configurations are supported by which tools. The first column, Obfuscation tool, shows our considered obfuscation tools; and the remaining five columns, all, opt, rename, cf, and se, show our considered configurations. A checkmark (√) indicates that a configuration is supported by a tool, and a blank space indicates that a configuration is not supported. As the table shows, there are 18 supported combinations. In the remainder of the paper, we will refer to a combination of an obfuscation tool and an obfuscation configuration as an obfuscation. To the best of our knowledge, the considered obfuscations are deterministic in that multiple applications of the obfuscation to the same application produce identical results.

2.5. Measurement platforms

To measure the amount of energy consumed by executing an Android application, we used four custom-built EMPs. Each EMP is based on a commercial Android smartphone platform: the first EMP is based on a Nexus 3 with 32 GB of storage running Android version 4.3 (Jelly Bean); the second EMP is based on a Nexus 4 with 8 GB of storage running Android version 4.3 (Jelly Bean); the third EMP is based on a Samsung Galaxy S II with 16 GB of storage running Android version 4.3 (Jelly Bean); and the fourth EMP is based on a Samsung Galaxy S5 with 16 GB of storage running Android version 4.4 (Kit Kat). Figure 1 shows pictures of the EMPs we built.

Instead of using the phone’s battery, the EMPs use an external source to power the devices. The Nexus 4 and Galaxy S5 EMPs use a 30-V, 5-A DC power supply (KORAD KA3005D), while the Nexus 3 and the Galaxy S II EMPs use a Monsoon Power Monitor from Monsoon Solutions Inc (https://www.msoon.com/LabEquipment/PowerMonitor/). In both cases, using an external power source ensures that the phone’s battery monitor senses a constant charge level and allows us to compare results across executions without having to worry about variations in the physical battery’s performance or the phone’s power-saving infrastructure.

To sample the voltage and current draw of the phone, the Nexus 4 and Galaxy S5 EMPs use two Arduino Unos, each equipped with an Adafruit INA219 High Side DC Current Sensor board. One Arduino is used to sense the voltage and current drawn from the DC power supply, and the other is used to sense the voltage and current draw over the phone’s USB port. The Nexus 3 and Galaxy S II EMPs use the Monsoon Power Monitor, which is equipped with a dual range, self-calibrating,
integrating system. It has two current ranges with a 16-bit analogue-to-digital converter, one with a high-resolution range and the other with a low-resolution range. Software continuously calibrates each of these and selects the proper range during measurement. Both the Arduino/INA219 and the Monsoon Power Monitor report voltage measurements in volts and current measurements in milliamps.

Because our EMPs measure power consumption via hardware that is external to the phones, they do not introduce any measurement overhead to the application. This is ideal, because it means that we do not have to factor out the amount of energy consumed by the monitoring infrastructure itself. However, it also means that the EMPs and the phones do not share a single clock that can be used to identify which samples occurred during an execution of interest. A desktop computer can solve this problem by providing the global clock necessary for performing synchronization. By having the desktop computer start the execution of interest over Android Debug Bridge, it is possible to discard power samples recorded before the start of the execution. Similarly, because the duration of the recorded scenarios is known, it is possible to identify samples that were recorded after the end of the execution.

While the EMP itself does not introduce measurement overhead, the replay infrastructure does—in order to replay a recorded execution, RERAN installs an application on the phone that injects events into the Android kernel’s device drivers. However, because the RERAN process spends most of its time sleeping—it only wakes up to inject events—its overhead is negligible. In addition, because we are concerned with energy usage relative to a base line (i.e., before and after applying an obfuscation) rather than absolute numbers, and the energy costs are consistent across executions; factoring out this cost is not necessary.

2.6. Procedure

Figure 2 shows, at a high level, the procedure we followed in our study, divided into four main steps: Obfuscated Application Creation, Replay-able Execution Creation, Data Collection, and Post-processing. The remainder of this section describes these steps in detail.

2.6.1. Obfuscated application creation. The first step in our procedure was to create our set of obfuscated applications. To create the necessary obfuscated versions, we obfuscated each application (Table I) using each obfuscation (Table III). In total, we created 198 obfuscated applications: 11 applications, each with 18 obfuscated versions.
2.6.2. Replay-able execution creation. The second step in our procedure was to create our set of replay-able executions. To create the replay-able executions, we manually performed the actions contained in each scenario while using RERAN’s recording tool. Because the replays produced by RERAN are not portable across mobile phone platforms, we created four replay-able executions for each scenario, one for each of our considered EMP platforms. This resulted in a total of 84 replay-able executions (21 scenarios × 4 platforms). As a sanity check, we then verified that RERAN could accurately replay each execution by running RERAN with the replay-able execution as input and observing the replayed executions. Table IV shows the durations of the replay-able executions for each scenario. The first two columns, Application and Name, show the scenario from Table II, and the remaining four columns, Nexus 3 through Galaxy S5, show the duration of the replay-able execution for each platform in seconds.

2.6.3. Data collection. The third step in our procedure was to collect power usage data. To collect power usage data, we used RERAN to replay each replay-able execution (Table IV) on the corresponding EMP, using both the unobfuscated and obfuscated versions of the scenario’s application. For each EMP, each replay-able execution was executed on each version of the

<table>
<thead>
<tr>
<th>Application</th>
<th>Name</th>
<th>Nexus 3</th>
<th>Nexus 4</th>
<th>Galaxy S II</th>
<th>Galaxy S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnkiDroid</td>
<td>New deck</td>
<td>87</td>
<td>128</td>
<td>83</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>Tutorial deck</td>
<td>64</td>
<td>60</td>
<td>108</td>
<td>95</td>
</tr>
<tr>
<td>Calculator</td>
<td>Advance</td>
<td>51</td>
<td>79</td>
<td>60</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>Standard</td>
<td>54</td>
<td>58</td>
<td>57</td>
<td>43</td>
</tr>
<tr>
<td>Calendar</td>
<td>Add event</td>
<td>108</td>
<td>108</td>
<td>113</td>
<td>104</td>
</tr>
<tr>
<td>Clock</td>
<td>Interval</td>
<td>28</td>
<td>65</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Stopwatch</td>
<td>18</td>
<td>20</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Timer</td>
<td>21</td>
<td>19</td>
<td>23</td>
<td>20</td>
</tr>
<tr>
<td>DailyMoney</td>
<td>Add detail</td>
<td>30</td>
<td>34</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>View lists</td>
<td>14</td>
<td>16</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>FrozenBubblePlus</td>
<td>Level 1</td>
<td>29</td>
<td>36</td>
<td>27</td>
<td>45</td>
</tr>
<tr>
<td>Nim</td>
<td>Easy AI</td>
<td>43</td>
<td>75</td>
<td>61</td>
<td>78</td>
</tr>
<tr>
<td>OIFileManage</td>
<td>Create file</td>
<td>46</td>
<td>60</td>
<td>80</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Play file</td>
<td>60</td>
<td>59</td>
<td>52</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>View file</td>
<td>22</td>
<td>18</td>
<td>22</td>
<td>36</td>
</tr>
<tr>
<td>OpenSudoku</td>
<td>Easy level 1</td>
<td>138</td>
<td>273</td>
<td>237</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td>Hard level 1</td>
<td>145</td>
<td>135</td>
<td>223</td>
<td>129</td>
</tr>
<tr>
<td>SkyMap</td>
<td>Find Mars</td>
<td>49</td>
<td>42</td>
<td>56</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Move zoom</td>
<td>21</td>
<td>65</td>
<td>19</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Show component</td>
<td>90</td>
<td>100</td>
<td>68</td>
<td>101</td>
</tr>
<tr>
<td>Tomdroid</td>
<td>Notes</td>
<td>72</td>
<td>173</td>
<td>62</td>
<td>131</td>
</tr>
</tbody>
</table>
application (unobfuscated and obfuscated) 30 times as is suggested by well-known guidelines for empirical study design [2]. While each scenario was executing, we recorded the current and voltage measurements using the EMP.

To reduce the possibility of noise in the measurements, we terminated all unnecessary applications and processes and, when possible, enabled airplane mode. Although we eliminated many possible sources of noise by carefully configuring the EMPs, small fluctuations in energy usage from execution to execution were still possible. For example, garbage collection or other operating-system level processes that could not be disabled may have been able to impact energy usage. Multiple runs (i.e., 30) allowed us to perform a statistical analysis on the impact of obfuscations that took into account the possibility of such fluctuations.

In total, we ran 47,880 executions—21 scenarios × (18 obfuscated versions + 1 unobfuscated version) × 30 repetitions × 4 EMPs—which took ≈924 h (over 5 weeks) of continuous execution time and resulted in over 15 GB of raw power consumption data.

2.6.4. Post-processing. The final step in our procedure was to post-process the collected data by filtering it and converting it to a usable form. We first filtered the data to remove samples that occurred either before or after the execution. We then converted the current and voltage samples to power measurements in watts by multiplying them together and then dividing by 1000: watts = volts × milliamperes ÷ 1000. Finally, we converted the resulting power measurements to total energy usage in joules by summing the results of multiplying each power measurement by the length of time between itself and the following sample: joules = watts × seconds.

3. DATA ANALYSIS AND DISCUSSION

We refined our overall question of whether or not applying obfuscations can impact the energy usage of an application into the following specific research questions:

- **RQ1**—Impact. Do obfuscations impact the energy usage of an application? If so, how?
- **RQ2**—Consistency. Are there any significant differences in the impacts of the considered obfuscation tools or the considered obfuscation configurations?
- **RQ3**—Importance. Are the impacts of applying obfuscations likely to be meaningful or noticeable to a typical mobile application user?

The remainder of this section discusses the results of our study in terms of these research questions. Note that in answering these questions, we are only analyzing the data for each platform separately. Because the replay-able executions are not identical (Section 2.6.2), it would be inappropriate to analyze the impacts of the obfuscations across platforms.

3.1. **RQ1: Impact**

To gather the data necessary to answer our first research question, we performed Mann–Whitney–Wilcoxon (wilcox) tests to determine whether the difference between the amount of energy consumed by each scenario when run using the unobfuscated version of the application and each obfuscated version of the application is statistically significant. To check for statistical significance, we chose to use the Mann–Whitney–Wilcoxon test because we have one nominal variable (the obfuscation applied to the application) and one measurement value (the amount of energy consumed by the execution), and the test does not require that the data be normally distributed. The resulting p values were adjusted using Benjamini and Hochberg’s false discovery rate controlling method to account for performing multiple comparisons [6]. We chose an α of 0.05 and used R version 3.0.3’s implementation of the test (i.e., wilcox.test). Of the 1512 tests that we conducted, 378 (21 scenarios × 18 obfuscations) for each of our four platforms, 791 (≈52%) indicated a statistically significant difference in the amount of energy consumed by the unobfuscated and obfuscated versions. For each platform, the number of statistically significant differences was 229 (≈61%) for the Nexus 3, 282 (≈75%) for the Nexus 4, 107 (≈28%) for the Galaxy S II, and 173 (≈46%) for the Galaxy S5.
For the cases where there is a statistically significant difference (i.e., $p \leq 0.05$), we computed Vargha and Delaney’s $\hat{A}_{12}$ statistic\(^5\) to calculate the size of the effect of applying the obfuscation [7]. In general, the $\hat{A}_{12}$ statistic ranges from 0 to 1 and indicates, on average, how often one technique outperforms another: when $\hat{A}_{12}$ is exactly 0.5, the two techniques achieve equal performance; when $\hat{A}_{12}$ is less than 0.5, the first technique performs worse; and when $\hat{A}_{12}$ is greater than 0.5, the second technique is worse. The closer $\hat{A}_{12}$ is to 0 or 1, larger the effect. For our data, $\hat{A}_{12}$ represents the probability that the unobfuscated version consumes more energy than the obfuscated version.

Figure 3a–d shows the $\hat{A}_{12}$ statistics we calculated. In the figures, the y-axis shows the considered scenarios, and the x-axis shows each obfuscation (combination of obfuscation tool and obfuscation configuration). For example, the first grouping shows the $\hat{A}_{12}$ statistics computed between the unobfuscated version of each application and the obfuscated versions produced by each obfuscation tool when using the all configuration. The color of each cell indicates the size and direction of the effect. Cells colored blue indicate cases where the unobfuscated version is more likely to consume more energy than the obfuscated version (i.e., $\hat{A}_{12} > 0.5$), and cells that are colored red indicate cases where the unobfuscated version is more likely to consume less energy than the obfuscated version (i.e., $\hat{A}_{12} < 0.5$). In addition, the color’s saturation indicates the size of the effect with the highest saturation indicating a large effect ($\hat{A}_{12}$ between 0.75 and 1.0 or between 0 and 0.25), a medium effect ($\hat{A}_{12}$ between 0.66 and 0.75 or between 0.25 and 0.33), or a small effect ($\hat{A}_{12}$ between 0.5 and 0.66 or between 0.33 and 0.5). Absent values indicate cases where there is not a statistically significant difference in energy usage between the versions.

From this data, we observe that, when all platforms are considered, obfuscations have a generally negative impact on energy usage (i.e., they increase energy usage). In 496 out of the 791 cases when there is a statistically significant difference in energy usage ($\approx 63\%$ of the time), the obfuscated version is more likely to consume more energy than the unobfuscated version. In the remaining 295 cases ($\approx 37\%$ of the time), the obfuscated version is more likely to consume less energy than the unobfuscated version. In addition, the size of the effect is most often large: the effect size is large for 575 cases ($\approx 73\%$ of the time), medium for 204 cases ($26\%$ of the time), and small for 12 cases ($\approx 1\%$ of the time).

When considered individually, obfuscations also have a negative impact for applications that are executed on the Nexus 4 and Galaxy S5. In 235 out of the 282 cases ($\approx 83\%$ of the time) for the Nexus 4 and 127 out of the 173 cases ($\approx 73\%$ of the time) for the Galaxy S5, when there is a statistically significant difference in energy usage, the obfuscated version is more likely to consume more energy than the unobfuscated version. For applications that are executed on the Galaxy S II, the obfuscations have a more balanced impact. For only 55 out of the 107 cases ($\approx 51\%$ of the time), the obfuscated version is more likely to consume more energy than the unobfuscated version. Finally, for applications that are executed on the Nexus 3, obfuscations have a more positive impact. For 150 out of the 229 cases ($\approx 66\%$ of the time), the obfuscated version is more likely to consume less energy than the unobfuscated version.

Next, we investigated the magnitude of the differences caused by the obfuscations. To determine the magnitude of the differences, we again focused on the cases where there is a significant difference in energy usage. For each combination of user scenario and obfuscation, we calculated the percentage change in mean of the energy usage between the obfuscated and unobfuscated versions. The results of these computations are shown in Figure 4a–d. The y-axis shows the usage scenarios, and the x-axis shows the obfuscations. The content of each cell shows the percentage change in mean energy usage. Again, the color of each cell indicates the direction and magnitude of the change. Blue cells indicate cases where the percentage change is negative (i.e., energy usage decreased), and red cells indicate cases where the percentage change is positive (i.e., energy usage increased); darker colors indicate larger values, and absent values indicate cases where there is not a statistically significant difference in energy usage.

Across all platforms, the percentage change in mean energy usage ranges from approximately equal to $-10.1\%$ to $\approx 6.9\%$ with a median and mean value of $\approx 0.5\%$ and a standard deviation of $\approx 2.1\%$.

\(^5\)Vargha and Delaney’s $\hat{A}_{12}$ statistic is a simple linear transformation of Cliff’s $d$: $\hat{A}_{12} = (d + 1)/2$. We prefer $\hat{A}_{12}$ because it is in the interval [0, 1], while $d$ is in the interval $[-1, 1]$. Eliminating the negative sign makes Figure 3 more readable.
For the Nexus 3, the percentage change in mean energy usage ranges from approximately equal to $\approx 10.1\%$ to $\approx 3.2\%$ with a median value of $\approx 0.7\%$, a mean value of $\approx 1.1\%$, and a standard deviation of $\approx 2.2$ percentage points.

For the Nexus 4, the percentage change in mean energy usage ranges from approximately equal to $\approx 3.7\%$ to $\approx 6.6\%$ with a median value of $\approx 1.2\%$, a mean value of $\approx 1.5\%$, and a standard deviation of $\approx 1.6$ percentage points. For the Galaxy S II, the percentage change in mean energy usage ranges from approximately equal to $\approx 5.5\%$ to $\approx 5.5\%$ with a median value of $\approx 0.2\%$, a mean value of $\approx 1.1\%$.

Figure 3. Vargha and Delaney’s $\hat{A}_{12}$—probability that an unobfuscated version consumes more energy than an obfuscated version when run on the Nexus 3 platform (wilcox, $p \leq 0.05$).

Figure 3. Vargha and Delaney’s $\hat{A}_{12}$—probability that an unobfuscated version consumes more energy than an obfuscated version when run on the Nexus 4 platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$).

Figure 3. Vargha and Delaney’s $\hat{A}_{12}$—probability that an unobfuscated version consumes more energy than an obfuscated version when run on the Galaxy S II platform (wilcox, Benjamini and Hochberg correction, $p \leq 0.05$).
c. Vargha and Delaney’s $\hat{A}_{12}$—probability that an unobfuscated version consumes more energy than an obfuscated version when run on the Galaxy S II platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$).

d. Vargha and Delaney’s $\hat{A}_{12}$—probability that an unobfuscated version consumes more energy than an obfuscated version when run on the Galaxy S5 platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$).

Figure 3. Continued

≈0.01%, and a standard deviation of ≈2.0 percentage points. For the Galaxy S5, the percentage change in mean energy usage ranges from approximately equal to −1.6% to ≈6.9% with a median value of ≈1.2%, a mean value of ≈1.2%, and a standard deviation of ≈1.6 percentage points.

From these data, it is clear that, while overall obfuscations are more likely to cause an increase in energy usage than a decrease in energy usage, the magnitude of the change, regardless of direction, is likely to be less than 5%. When compared with the energy impacts of other code level changes, the energy impacts of obfuscations are closer to the impacts of other focused changes (e.g., refactorings, whose impacts range from −7.50% to 4.54% [8] and switching application programming interface (API) implementations [9]) than to the impacts of more broad changes (e.g., applying design patterns, whose impacts can approach several hundred percent [10]).
Based on our investigations into the impacts of obfuscations on energy usage, we have found the following:

1. Obfuscations can, and often do, impact the energy usage of an application with statistical significance.
2. Individually, all of our considered obfuscation tools and obfuscation configurations can both increase and decrease energy usage.

Figure 4. Percent change in mean energy usage when using an obfuscated version instead of an unobfuscated version when run on the a. Nexus 3 platform (wilcoxon, Benjamini & Hochberg correction, $p \leq 0.05$).

<table>
<thead>
<tr>
<th>Obfuscation Tool</th>
<th>all</th>
<th>opt</th>
<th>rename</th>
<th>cf</th>
<th>se</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextSize New Dex</td>
<td>-0.69</td>
<td>-0.52</td>
<td>0.72</td>
<td>-0.67</td>
<td>0.67</td>
</tr>
<tr>
<td>TextSize Tutorial Dex</td>
<td>-1.22</td>
<td>-2.74</td>
<td>-2.18</td>
<td>-1.36</td>
<td>-2.13</td>
</tr>
<tr>
<td>Calculator</td>
<td>0.71</td>
<td>0.72</td>
<td>-0.89</td>
<td>0.77</td>
<td>0.77</td>
</tr>
<tr>
<td>Calculator: Add Event</td>
<td>0.68</td>
<td>0.69</td>
<td>-0.95</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>ClockInput</td>
<td>0.99</td>
<td>0.76</td>
<td>0.53</td>
<td>0.76</td>
<td>0.76</td>
</tr>
<tr>
<td>Clock: SleepEvents</td>
<td>-1.07</td>
<td>-0.67</td>
<td>-1.51</td>
<td>-0.67</td>
<td>-1.51</td>
</tr>
<tr>
<td>DailyMoney: Add Data</td>
<td>0.27</td>
<td>0.87</td>
<td>0.18</td>
<td>0.27</td>
<td>0.18</td>
</tr>
<tr>
<td>DailyMoney: View Data</td>
<td>-0.68</td>
<td>-0.62</td>
<td>-0.95</td>
<td>-0.68</td>
<td>-0.62</td>
</tr>
<tr>
<td>ProcExeBdd nuisance 1 &amp;</td>
<td>0.84</td>
<td>0.84</td>
<td>0.84</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>New: Easy AI</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td>OffSetManager: Create File</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>OffSetManager: Move File</td>
<td>0.54</td>
<td>0.54</td>
<td>0.54</td>
<td>0.54</td>
<td>0.54</td>
</tr>
<tr>
<td>Open: Daytime Easy 1 &amp;</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
</tr>
<tr>
<td>Open: Daytime Hard 1 &amp;</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
<td>-0.25</td>
</tr>
<tr>
<td>SkyMap: Show Component</td>
<td>1.07</td>
<td>1.07</td>
<td>1.07</td>
<td>1.07</td>
<td>1.07</td>
</tr>
<tr>
<td>Terminat: Notes</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
<td>0.37</td>
</tr>
</tbody>
</table>

a. Percent change in mean energy usage when using an obfuscated version instead of an unobfuscated version when run on the Nexus 3 platform (wilcoxon, $p \leq 0.05$).

b. Percent change in mean energy usage when using an obfuscated version instead of an unobfuscated version when run on the Nexus 4 platform (wilcoxon, Benjamini & Hochberg correction, $p \leq 0.05$).
Across all platforms, the likelihood of causing an increase in energy usage is higher than the likelihood of causing a decrease in energy usage.

Across all platforms, the magnitude of the percentage change in energy usage is most likely to be less than 5%.

### 3.2. RQ2: Consistency

The goal of our second research question is to determine if there is a statistically significant benefit, with respect to energy usage, to using a specific obfuscation tool or specific obfuscation configuration. To answer this question, we performed several Kruskal–Wallis tests. We chose to use the Wilcoxon, Benjamini & Hochberg correction, \( p \leq 0.05 \).

#### c. Percent change in mean energy usage when using an obfuscated version instead of an unobfuscated version when run on the Galaxy S II platform

<table>
<thead>
<tr>
<th>Usage Scenario</th>
<th>all</th>
<th>opt</th>
<th>rename</th>
<th>cf</th>
<th>se</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>-0.78</td>
<td>-0.62</td>
<td>-0.62</td>
<td>-0.63</td>
<td>-0.65</td>
</tr>
<tr>
<td>opt</td>
<td>-0.28</td>
<td>-0.05</td>
<td>-0.05</td>
<td>-0.05</td>
<td>-0.05</td>
</tr>
<tr>
<td>rename</td>
<td>-0.66</td>
<td>-0.66</td>
<td>-0.66</td>
<td>-0.66</td>
<td>-0.66</td>
</tr>
<tr>
<td>cf</td>
<td>1.00</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>se</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
</tr>
</tbody>
</table>

#### d. Percent change in mean energy usage when using an obfuscated version instead of an unobfuscated version when run on the Galaxy S5 platform

<table>
<thead>
<tr>
<th>Usage Scenario</th>
<th>all</th>
<th>opt</th>
<th>rename</th>
<th>cf</th>
<th>se</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>-0.74</td>
<td>-0.63</td>
<td>-0.63</td>
<td>-0.65</td>
<td>-0.65</td>
</tr>
<tr>
<td>opt</td>
<td>-0.41</td>
<td>-0.39</td>
<td>-0.39</td>
<td>-0.40</td>
<td>-0.40</td>
</tr>
<tr>
<td>rename</td>
<td>-0.41</td>
<td>-0.41</td>
<td>-0.41</td>
<td>-0.41</td>
<td>-0.41</td>
</tr>
<tr>
<td>cf</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>se</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Figure 4. Continued

(3) Across all platforms, the likelihood of causing an increase in energy usage is higher than the likelihood of causing a decrease in energy usage.

(4) Across all platforms, the magnitude of the percentage change in energy usage is most likely to be less than 5%.

3.2. RQ2: Consistency

The goal of our second research question is to determine if there is a statistically significant benefit, with respect to energy usage, to using a specific obfuscation tool or specific obfuscation configuration. To answer this question, we performed several Kruskal–Wallis tests. We chose to use
the Kruskal-Wallis test because we want to compare one measurement value (the amount of energy consumed by the execution) across multiple samples (obfuscation tools or obfuscation configurations), and we do not know if our data are normally distributed. We chose an $\alpha$ of 0.05 and used R version 3.1.2’s implementation of the test (i.e., kruskal.test). In general, if the $p$ value calculated by the Kruskal-Wallis test is less than the chosen $\alpha$, it indicates that at least one of the samples is significantly different from the others. It does not indicate how many differences occur or among which samples the differences exist. However, this information can be determined by running pairwise Mann-Whitney-Wilcoxon tests with an appropriate correction for performing multiple comparisons (e.g., Bonferroni correction, and Benjamini and Hochberg correction).

Our first set of Kruskal-Wallis tests checks whether there are any statistically significant differences in the percentage changes in mean energy usage among obfuscation tools for each obfuscation configuration. A $p$ value less than our chosen alpha would indicate that one of the obfuscation configurations is statistically different from the others. The results of these computations can be seen in Table V. In this table, the first column, Obfuscation Configuration, shows the name of each obfuscation configuration. The next four columns, Nexus 3 through Galaxy S5, show the $p$ value when each platform is considered individually; and the final column, All, shows the $p$ value when all four platforms are considered together. Because the computed $p$ values are never less than our chosen $\alpha$ (0.05), we cannot reject the null hypothesis. In practice, this means that, with respect to energy usage, there is no statistical benefit to picking one obfuscation tool over another. Consequently, developers are free to choose their preferred obfuscation tool based on other factors, such as supported obfuscations, price, and ease of use, without having to worry about its impact on energy usage.

Our second set of Kruskal-Wallis tests check whether there are any statistically significant differences in the percentage changes in mean energy usage among the obfuscation configurations for each obfuscation tool. The result of these computations can be seen in Table VI. The format of the table is similar to that of Table V. The first column, Obfuscation Tool, shows the name of each obfuscation tool. The remaining columns show the $p$ value when each platform is considered individually, Nexus 3 through Galaxy S5, and together, All. Again, because the computed $p$ values are never less than our chosen $\alpha$ (0.05), we cannot reject the null hypothesis. In practice, this means that, with respect to energy usage, there is no statistical benefit to picking one obfuscation configuration over another. Again, application developers are free to choose their preferred obfuscation configuration based on factors other than its impact on energy usage.

### 3.3. RQ3: Importance

Our first two research questions were primarily concerned with discovering if and how obfuscations impact the energy usage of applications. The goal of our third research question is to assess whether the observed impacts are likely to be meaningful or noticeable to mobile application users.

To answer this question, we first used Equation (1) to calculate, for each platform, the percentage of battery charge that is consumed by each scenario when it is executed using the unobfuscated version of its application and when it is executed using the obfuscated versions of its application.

$$\%\text{charge} = \frac{E}{V} \times \frac{1000}{C \times 3600} \times 100$$  \hspace{1cm} (1)

Table V. For an obfuscation configuration, is there a statistically significant difference among the obfuscation tools ($%\text{change} ~ \text{tool}$)?

<table>
<thead>
<tr>
<th>Obfuscation configuration</th>
<th>p value</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nexus 3</td>
<td>Nexus 4</td>
<td>Galaxy S II</td>
<td>Galaxy S5</td>
<td>All</td>
</tr>
<tr>
<td>all</td>
<td>0.39</td>
<td>0.56</td>
<td>0.18</td>
<td>0.45</td>
<td>0.08</td>
</tr>
<tr>
<td>opt</td>
<td>0.51</td>
<td>0.92</td>
<td>0.20</td>
<td>0.38</td>
<td>0.56</td>
</tr>
<tr>
<td>rename</td>
<td>0.56</td>
<td>0.75</td>
<td>0.56</td>
<td>0.91</td>
<td>0.83</td>
</tr>
<tr>
<td>cf</td>
<td>0.67</td>
<td>0.90</td>
<td>0.89</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>se</td>
<td>0.80</td>
<td>0.52</td>
<td>0.92</td>
<td>0.87</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Copyright © 2016 John Wiley & Sons, Ltd.
In Equation (1), $E$ is the amount of energy in joules consumed by an execution (here, we used the mean energy usage of each version of our 30 trials), $V$ is the output voltage of the platform’s battery in volts, and $C$ is the electric charge of the platform’s battery in milliampere hours. For the Nexus 3, $V=3.7\, \text{V}$ and $C=1900\, \text{mAh}$; for the Nexus 4, $V=3.8\, \text{V}$ and $C=2100\, \text{mAh}$; for the Galaxy S II $V=3.6\, \text{V}$ and $C=1800\, \text{mAh}$; and for the Galaxy S5 $V=3.8\, \text{V}$ and $C=2800\, \text{mAh}$.

We then calculated, using Equation (2), the amount of time needed to drain Nexus 4’s battery from full to empty (i.e., battery life) if the scenario were executed continuously using each version of its application.

$$t_{\text{drain}} = \frac{100\%}{\%\text{charge}} \times D$$

(2)

In Equation (2), $\%\text{charge}$ is the percentage of battery charge calculated using Equation (1) and $D$ is the duration of the scenario (Table IV). Note that the unit of measurement for $t_{\text{drain}}$ will be the same as the unit of measurement for $D$.

Table VII shows the results of this computation. In the table, the first two columns, Application and Name, show the scenario; and the remaining columns, Nexus 3 through Galaxy S5, show, for each platform, the mean battery life in hours when the unobfuscated version is run continuously, draining the battery from full to empty.

Table VII. Battery life when using an unobfuscated version.
Finally, we computed the change in battery life for each scenario and obfuscation by subtracting the battery life of each obfuscated version from the battery life of the unobfuscated version. Figure 5a–d shows the results of these computations. The five groupings in each figure show the change in mean battery life in minutes when an obfuscated version is used instead of an unobfuscated version. Again, absent values indicate instances where there was no statistically significant difference in energy usage between the application versions, and the color of each cell indicates the direction and magnitude of the change. Blue cells indicate obfuscations that

---

### Figure 5. Change in mean battery life when using an obfuscated version instead of an unobfuscated version when run on the (a) Nexus 3 platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$); (b) Nexus 4 platform (wilcox, Benjamini and Hochberg correction, $p \leq 0.05$); (c) Galaxy S II platform (wilcox, Benjamini and Hochberg correction, $p \leq 0.05$); and (d) Galaxy S5 platform (wilcox, Benjamini and Hochberg correction, $p \leq 0.05$).

**a. Change in mean battery life when using an obfuscated version instead of an unobfuscated version when run on the Nexus 3 platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$).**

**b. Change in mean battery life when using an obfuscated version instead of an unobfuscated version when run on the Nexus 4 platform (wilcox, Benjamini & Hochberg correction, $p \leq 0.05$).**
increase battery life (i.e., changes that are beneficial for users), and red cells indicate obfuscations that decrease battery life (i.e., changes that are detrimental to users).

Across all configurations, the change in battery life for the Nexus 3 ranges from approximately equal to $-8.4$ to $\approx 22.0$ min with a mean value of $\approx 2.5$ min, a median value of $\approx 1.9$ min, and a standard deviation of $\approx 4.6$ min. The change in battery life for the Nexus 4 ranges from approximately equal to $-16.3$ to $\approx 10.9$ min with a mean value of approximately equal to $-3.9$ min, a median value of approximately equal to $-3.1$ min, and a standard deviation of $\approx 4.2$ min. The change in battery life for the Galaxy S II ranges from approximately equal to $-13.9$ to $\approx 11.5$ min with a mean value of approximately equal to $-1.1$ min, a median value of approximately equal to $-0.9$ min, and a standard deviation of $\approx 5.6$ min and the change in battery life for the Galaxy S5 ranges from...
When only the All configuration is considered, the change in battery life for the Nexus 3 ranges from approximately equal to \(-3.3\) to \(\approx 14.1\) min with a mean value of \(\approx 2.4\) min, a median value of \(\approx 2.1\) min, and a standard deviation of \(\approx 3.9\) min. The change in battery life for the Nexus 4 ranges from approximately equal to \(-14.9\) to \(\approx 2.9\) min with a mean value of \(\approx 4.6\) min, a median value of \(\approx 4.9\) min, and a standard deviation of \(\approx 7.0\) min.

Based on these results, we believe that it is unlikely for an application user to notice a decrease in battery life due to an obfuscation. The observed changes in battery life range from approximately equal to \(-31.4\) to \(\approx 22.0\) min, which, even for the maximum and minimum, represents a change of less than 10% of the respective phone’s total battery life. Recall that these are the expected changes if the scenarios were executed continuously, draining the battery from full to empty. In practice, this is unlikely because mobile phone users rarely use an application continuously.

In retrospect, this result makes sense. For mobile applications, recent studies show that the majority of energy is consumed by the phone’s screen, radios, and sensors [11]. The changes made by the obfuscations do not change how the applications interact with or use these resources. Because the obfuscations make changes to parts of the application that do not consume much energy, the impacts of the obfuscations are overshadowed by the more energy-expensive parts of the execution.

While users are likely to be indifferent to this conclusion because obfuscations neither harm nor improve their battery life, it is good news for application developers. Now, developers are able to protect their applications by applying obfuscations without needing to consider the obfuscation’s impacts on energy usage.

### 3.4. Threats to validity

One of the most significant threats to the validity of our results is the possibility of energy usage measurement errors due to either imprecise measurements or failing to control for potential sources of noise and nondeterminism. To minimize this threat, we took several steps. First, we designed and implemented our EMPs with the help of experienced electrical engineers from the Department of Electrical and Computer Engineering at the University of Delaware or used equipment that had been independently built and evaluated (i.e., Monsoon Power Monitor). Second, we offloaded the monitoring infrastructure to external hardware so that it could not add any overhead to the energy usage of the execution that was being measured. Third, we used an existing capture/replay tool to deterministically reproduce a recorded execution to eliminate biases in how the applications were executed. Finally, we terminated all unnecessary applications and processes and, when possible, enabled airplane mode, which helped reduce noise and sources of nondeterminism caused by external processes (e.g., garbage collection).

An additional threat to validity is that we considered only 11 applications. Although we selected these applications to cover different application types, it is possible that they may not be representative of all applications for reasons that would not be readily apparent from interacting with them. Similarly, considering only Android applications may prevent our conclusions from generalizing to all environments where obfuscations are used (e.g., Windows Phone applications and non-Android Java applications). In future work, we plan to address these threats by expanding our study to include additional applications and additional devices.

A more specific concern is that, owing to the limitations of RERAN, our set of considered applications does not contain applications that make heavy use of the network or sensors beyond the touch screen. However, although network connections and sensors are known to consume large amount of power, the changes made by the obfuscation tools will not affect how the applications
use these resources (i.e., the number of types of calls made to these APIs will not be changed). Consequently, we believe that the results we observed will also hold for such applications.

Finally, our choice of usage scenarios for driving each application may not be representative of how the applications are actually used in practice. However, we believe that this is unlikely because the scenarios were generated by using each application for its intended purposes.

4. RELATED WORK

The most closely related area of work is a recent group of papers that have attempted to identify the underlying causes of energy usage by empirically investigating the impact of various software development decisions. More specifically, researchers have investigated the impacts of refactorings [8, 12], advertisements [13], design patterns [10, 14, 15], sorting algorithms [16], test suite selection [17], web servers [18], programming models [19, 20], API usage [9, 21], and lock-free data structures [22] within a single application in addition to investigating trends in an application’s energy usage among versions [23] and among separate implementations of the same specification [24, 25]. In addition, researchers have investigated trends in an application’s energy usage among versions [23] and how developers ask questions about energy usage [26].

Another area of related work is techniques that attempt to detect energy bugs (e.g., [24, 27–30]). Although such techniques are effective at detecting certain types of energy bugs, it is not clear whether they would be able to detect unnecessary energy usage as a result of applying obfuscations. In general, these techniques can only detect bugs that result in large, abnormal spikes in energy usage or repeated increases in energy usage without a corresponding user action (e.g., polling the GPS when the screen is off). Applying an obfuscation is unlikely to cause either of these situations to occur. Rather, it is likely to increase the overall energy usage by a small, constant amount.

In addition to our work at the high level, there is a significant amount of research on optimizing energy usage at the programming language and system levels. At the programming language level, there are several approaches to helping developers write more energy-efficient software. Such work includes new type of systems (e.g., [31]), new programming models (e.g., [32–34]), mechanisms for exposing energy-expensive architectural details (e.g., [35, 36]), and manipulating quality of service [37] and the precision of the results of the computation at runtime [38, 39]. At the compiler level, work has focused on optimizing code to use fewer instructions or a more efficient ordering of instructions; controlling hibernation, dynamic frequency, and voltage scaling; and remote task mapping (e.g., [40–50]). At the operating-system level, work has focused on the goals of allowing an operating system to manage energy in the same manner as other system resources (e.g., [51]) and optimizing the balance between power and performance via the automatic selection of power policies during application execution (e.g., [52–54]). At the hardware level, work focused on reducing excessive CPU cycles (e.g., [55]), capping RAM energy usage (e.g., [56]), and the addition of special core to support common virtual machine operations (e.g., [57]). There is also significant work in the area of high-performance computing (e.g., [58–64]).

Kansal et al. created Senergy, an API to improve the energy efficiency of applications that use sensor data [65], while Liu et al. created a technique to save energy when mobile applications access but do not use sensor data. Computation offloading (to another computer or the cloud) can provide energy savings particularly useful for mobile devices as the battery is limited, and several tools have been developed for helping programmers indicate which computations to offload [66–69]. Li et al. created Nyx to rewrite the server-side code of web applications so that the resulting pages would consume less energy when displayed on a smartphone [70].

The ability to measure the energy usage of a piece of software is a necessary prerequisite for optimizing its energy usage. Work in the area of energy usage measurement has been conducted at various levels. Hardware instrumentation-based approaches (e.g., [71], https://www.wattsupmeters.com/secure/index.php) use physical instrumentation (i.e., soldering wires to power leads) to measure the actual power usage of a system. Such approaches have the benefit of being accurate because they measure actual power usage; however, they are also expensive and difficult to use because they require specialized hardware. Simulation-based approaches (e.g., [72–74]) use a cycle-accurate
simulator to replicate the actions of a processor at the architecture level and estimate energy usage of each executed cycle. Like hardware instrumentation-based approaches, simulation-based approaches can be accurate, but they are also difficult to use. Finally, estimation-based approaches (e.g., [20, 23, 75–82]) build models of energy-influencing features and use such models to estimate energy usage. For example, Hao et al. and Seo et al. construct energy models of Java bytecode and use the models to estimate the energy usage of a given method or execution path [20, 76, 77]. Estimation-based approaches are frequently less accurate than hardware instrumentation-based or simulation-based approaches, but they have the benefits of being more widely applicable and easier to use.

5. CONCLUSIONS AND FUTURE WORK

We have presented an empirical study that investigated the impact of code obfuscations on the energy usage of mobile applications. We considered 11 commonly used Android applications, four obfuscation tools, five obfuscation configurations, 21 usage scenarios, and four platforms. In total, we generated and ran over 47,000 executions on our EMPs.

The results of this study demonstrate the following:

1. Obfuscations can, and often do, impact the energy usage of applications with statistical significance.
2. Obfuscations can both increase and decrease energy usage, but they are more likely to increase energy usage.
3. The magnitude of the impacts of obfuscations is comparable with the magnitude of the impacts of other code level changes, such as applying refactorings.
4. The differences between the impacts of the considered obfuscations on energy usage are not statistically significant.
5. The impacts of obfuscation on battery life are unlikely to be meaningful to mobile application users.

As such, we believe that these results are positive news for both mobile application users and mobile application developers. Developers can protect their applications without impacting the battery life of the devices where their applications execute.

Based on these conclusions, there are several potential areas for future work.

First, we plan on enlarging the scope of our study. Although we considered a significant number of applications and obfuscations, adding additional applications, platforms (e.g., tablets), and architectures (e.g., Windows phone) would potentially allow us to confirm or refute our observations.

Second, we plan on investigating the underlying reasons for why the obfuscations exhibit the observed impacts and whether other types of information can be used to accurately predict the impacts of applying obfuscations. From the perspective of a software engineer, the answer to this question has a high utility. In most cases, developers do not have access to the type of custom, hardware-based energy profiling tools that we do. As a result, they have no way of identifying whether obfuscating their code has or will increase or decrease energy usage. Providing them with the ability to make accurate predictions about the impacts of applying obfuscations would be very useful.

ACKNOWLEDGEMENTS

This work is supported in part by National Science Foundation grant no. 1216488 and no. 1321141. We would also like to thank Smardec, PreEmptive Solutions, and Zelix Pty. Ltd. for providing full-featured copies of their obfuscation tools. Finally, we would like to thank Abram Hindle for his initial guidance on building the Nexus 4-based EMP.

REFERENCES


