Schrödinger's Security: Opening the Box on App Developers' Security Rationale

Dirk van der Linden
University of Bristol
dirk.vanderlinden@bristol.ac.uk

Pauline Anthonysamy
Google
anthonysp@google.com

Bashar Nuseibeh
The Open University
Lero, University of Limerick
b.nuseibeh@open.ac.uk

Thein Than Tun
The Open University
thein.tun@open.ac.uk

Marian Petre
The Open University
m.petre@open.ac.uk

Mark Levine
Lancaster University
mark.levine@lancaster.ac.uk

John Towse
Lancaster University
j.towse@lancaster.ac.uk

Awais Rashid
University of Bristol
awais.rashid@bristol.ac.uk

ABSTRACT

Research has established the wide variety of security failures in mobile apps, their consequences, and how app developers introduce or exacerbate them. What is not well known is *why* developers do so—what is the rationale underpinning the decisions they make which eventually strengthen or weaken app security? This is all the more complicated in modern app development's increasingly diverse demographic: growing numbers of independent, solo, or small team developers who do not have the organizational structures and support that larger software development houses enjoy.

Through two studies, *we open the box on developer rationale*, by performing a holistic analysis of the rationale underpinning various activities in which app developers engage when developing an app.

The first study does so through a task-based study with app developers (N=44) incorporating six distinct tasks for which this developer demographic must take responsibility: setting up a development environment, reviewing code, seeking help, seeking testers, selecting an advertisement SDK, and software licensing. We found that, while on first glance in several activities participants seemed to prioritize security, only in the code task such prioritization was underpinned by a security rationale–indicating that development behavior perceived to be secure may only be an illusion until the box is opened on their rationale.

The second study confirms these findings through a wider survey of app developers (N=274) investigating to what extent they find the activities of the task-based study to affect their app's security. In line with the task-based study, we found that developers perceived *actively* writing code and *actively* using external SDKs as the only security-relevant, while similarly disregarding other activities having an impact on app security.

Our results suggest the need for a stronger focus on the tasks and activities *surrounding* the coding task – all of which need to be underpinned by a security rationale. Without such a holistic focus, developers may *write* "secure code" but not *produce* "secure apps".

CCS CONCEPTS

• **Security and privacy → Social aspects of security and privacy**.

ACM Reference Format:
Dirk van der Linden, Pauline Anthonysamy, Bashar Nuseibeh, Thein Than Tun, Marian Petre, Mark Levine, John Towse, and Awais Rashid. 2020. Schrödinger's Security: Opening the Box on App Developers' Security Rationale. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380394

1 INTRODUCTION

Security failures in mobile apps and their consequences are well understood – ranging from forgetting to sanitize user input to third party libraries using dynamic code-loading to execute malicious content [9, 19, 21, 25], or simply copy pasting insecure or obsolete code snippets [22]. But, secure app development is about more than just writing secure code. A range of activities, such as, choice of development environment plugins, seeking help when things go wrong and monetization of the app via third party ad libraries, all have a potential impact on security. Prior work has focused on what developers do—although primarily in activities that surround the actual writing of code—we instead ask *what drives developers to do things securely or not across the variety of app development activities*: *what rationale underpins their decisions?*

To complicate matters, *the demographic of app developers has expanded significantly as well*: app development is no longer the domain of the select few with deep technical skills, training and knowledge. Apps are now being developed by a wide range of people with diverse backgrounds. Already in 2012 reports showed that 40% of app developers were independent solo developers [17],

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380394

and more recent reports have shown that solo *hobbyist* and *amateur* developers constitute 43% of the app developer community. Professional app developers' have moved towards smaller scales as well: solo or small scale developer teams constitute a further 36% of the demographic [42]. With little ability to delegate things these (often independent) developers do not know how to do – or do not want to do – they must tackle a number of activities and considerations [18], exemplified in Figure 1. What developers do during all these activities (and not just while writing code) – and the rationale underpinning their actions – may compromise or worsen the security of the resulting app.

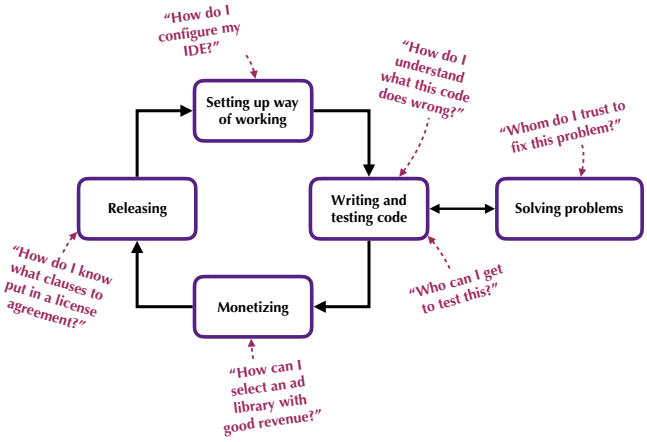


Figure 1: App development activities, with typical considerations developers run across, adapted from [18, 38].

Yet, this increasingly diverse and independent group of developers remains under-studied, much research (e.g., [6, 7, 33]) focusing on experienced, well-established developers. Research also tended to focus more on *what* established developers do (wrong), e.g., programming errors [21], misuse of either APIs [33] or online information sources (e.g., Stackoverflow [3]) – while not eliciting *why* they do so. For instance, researchers have shown that copy/pasting from StackOverflow may be harmful [22]. There are reasons *why* developers do it: shrinking deadlines, frustration with code not compiling, poor documentation – all contribute toward a valid rationale for developers to 'throw caution to the wind' and do something they may very well know to be less secure.

Similarly, while developers may understand the potential security implications of allowing third-party code to load into their app and execute, the necessary step of monetization brings exactly this threat of dynamic code loading. With developers often failing to convince users to pay even $0.99 for an app (let alone consider subscription fees for a service), many turn to advertisements as a reliable way of income. An increasing number of these libraries turn out to be malicious, e.g., Xavier [20] and Igexin [10]. Likewise, most people would intuitively avoid accepting gifts from strangers. Or, in the context of app development, a USB stick from a local user group with a copy of one's preferred platform development tools. Yet, as the Xcode-ghost malware that affected > 500 million iOS users [39] showed, there was a valid rationale behind doing exactly that: many developers lived in areas where they did not have sufficient bandwidth to (easily) download Xcode.

This paper is the first to present empirical insight into the rationale underpinning the various activities in which the *ever-growing part of the developer community of diverse and independent app developers* engage (cf. Fig. 1) and the extent to which security features in this rationale. The insight is based on two studies of app developers who have gone through such activities and published apps:

(1) An in-depth exploration of developers' prioritization (and subsequent rationalization) during tasks modeling these activities—*what they do securely or insecurely, and why.*
(2) A survey of a large sample of app developers' attitude towards the impact of various activities on app security—*whether a wider representative sample of app developers' attitudes supports or rejects the in-depth exploration.*

Our analysis is the first to uncover that while developers may often make secure decisions across the different activities they have to deal with—even activities not dealing with code—*opening the box on their rationale collapses the illusion of developers knowingly and intentionally acting securely.* We contribute the following:

- When faced with a coding task, almost half (43%) of participants rationalize their decision using security considerations. However, when faced with any other kind of task, developers rarely rationalize their decisions using security considerations (supported by binomial test comparison of rationale proportions, $p<0.01$). This indicates that developers may only really consider security when facing code.
- When faced with most other tasks (e.g., building a license agreement), half (50%) of the participants performed them securely but did so *without* expressing a clear security rationale. This indicates that perceived secure development behavior may only be an illusion.
- When faced with the two tasks that involve (the need for) social interaction with others, few (14% and 23%) participants performed them securely (supported by binomial test comparison of task solutions, $p<0.01$). This indicates that developers may underestimate the impact their social interaction with other people can have on their app's security.
- A wider survey of a representative app developer sample (N=274) further supports the finding that security rationales are focused on code-centric activities and perceived secure behavior across the wider spectrum of activities may be an illusion: while writing of code and active use of external SDKs are perceived to affect an app's security, no other activities are clearly perceived by developers as impacting security.

These results highlight that, while code vulnerabilities are the genesis of software insecurity, *focusing solely on one's code is not a foolproof strategy to avoid introducing or exacerbating vulnerabilities.* Developers need to be supported in understanding the security implications of their actions when deciding upon their way of working, trusting third party resources, interacting with third parties, and thinking of longer-term impacts of licensing. Researchers, equally, should provide support, whether tools or methods, that focus not only on code-level support, but non-obtrusively support secure reasoning and decision making in the above activities.

The rest of this paper proceeds as follows: Section 2 reviews and contrasts related work. Section 3 describes the design of our experiment, and Section 4 reports its results. Section 5 reports the design of our wider survey of app developers, and Section 6 reports it results. We discuss potential limitations and biases in Section 7 and conclude in Section 9.

2 BACKGROUND AND RELATED WORK

2.1 Vulnerabilities in mobile apps

Security failures in mobile apps and their consequences – ranging from developers forgetting to sanitize user input to third-party libraries using dynamic code-loading to execute malicious content – are well understood [9, 19, 21, 25]. The factors that influence security on major mobile ecosystems such as Android and iOS [1, 30], as well as attackers' behavior to some extent [8], have also been studied. On the developer side, vulnerability mitigation typically focuses on tool-based interventions to catch vulnerabilities through static or dynamic analysis (cf. [36, 37, 40]), assess third-party code or binaries loaded into the app [35], and the promotion of APIs which take critical code out of developers' hands [41], the usability of which for developers has been criticized [1].

Our focus on understanding what app developers prioritize and the rationale they give for their decisions complements this body of work by highlighting *why* vulnerabilities arise. This, in turn, can help to focus interventions not just on identifying vulnerabilities in code, but also on other tasks surrounding coding where the root causes may lie.

2.2 Developer motivation

A 2012 study of mobile app developers found that 40% worked as independents, 27% in small 2–3 person organizations [17]. Recent professional reports [43] show that app developers have less professional experience, and are not as motivated by financial gains as others. For example, the solo 'hobbyists' (15%) and 'explorers' (28%) – those working to gain experience, constituted almost half of the developer demographic in 2016. Other demographics constitute a rather significant proportion, such as 'hunters': independent developers focused on the money (21%), and 'guns for hire': independent developers working on commission (15%). *These demographics clearly emphasize the need to understand the reality of this new group of less established developers.* When it comes to motivations for security, developers vary to what extent they consider it to be part of their development process [7]. While many developers are known to pass responsibility for security to others if they can [49] and focus instead on functionality [47], the growing demographic of independent app developers may not have this luxury, or have to rely on different means to have recourse on others' knowledge. Intrinsic motivation for secure software has been found to translate into better attitude toward secure development [5]. An important anti-motivation in the context of app development is a perceived lack of self-competence, which arises through a lack of extant resources and support [5], which may arise especially in the case of less experienced app developers working independently.

Our work shows that, notwithstanding code, by and large developers do not reason about security in many development activities—providing a contrasting viewpoint that it is not so much whether developers are (de)motivated to work securely, but that it may never even occur to them.

2.3 Developer-centred security

Recent work has started to focus on developer-centred security, both understanding (e.g., [7, 27, 48]) and improving (e.g., [28, 46]) how they engage with security. Much of this work is focused on security surrounding technical code activities (e.g., coding, testing, deploying), or studying developers in traditional larger software development organizations (e.g., [6, 24]). Research has also shown that security knowledge does not guarantee secure software: some developers with good understanding of security practices fail to apply it, while other developers with very little security knowledge somehow built secure software [31].

Little work has yet focused on this breadth of knowledge now required of developers. For example, a recent survey of security advice most visible to developers [4] discusses guidance centered almost exclusively on code-related security concerns. A recent research agenda similarly focuses on coding activities [3], finding wildly differing attitudes towards security in this task alone.

In contrast our findings offer support for the position that, especially in the app development context characterized by individuals and small teams, security vulnerabilities should be understood in context of the rationale for the activities that introduced them. Only by understanding the lack of security rationale can we begin to conceptualize interventions to mitigate these vulnerabilities.

3 STUDY I: SECURITY RATIONALES ACROSS APP DEVELOPMENT ACTIVITIES

We designed a task-based online study to address three research questions:

(1) Do app developers prioritize security during different activities in which they engage while producing the app?
(2) Are activities where security is prioritized informed by a security rationale?
(3) What explanations underlie the rationales that developers have for their prioritization?

Drawing on examples established in the literature, we identified six tasks that reflect the activities that developers typically undertake [18], all of which present some form of security concern. We provided brief scenarios, and a selection of choices – some security focused, and some not. An overview of the tasks as they were presented to participants – and how we operationalized whether participants' selection and sorting of choices prioritizes security or not, is given in Fig. 2.

In order to make the study tractable online we based the tasks directly on activities found in recent literature summarizing the wide spectrum of activities in software development [18], taking care to ground them in relevant literature showing the effect of insecure decisions in licensing [12, 14], incorporation of advertisement SDKs [11, 44], seeking out of testers [16, 26], and use of different sources for help [3, 29].

Tasks 1 and 2 were selection tasks (see Fig. 2), in which participants had to make a single choice while viewing source-code (Task 2), or choose a maximum of 3 from a number of possible options

Task 1: Setting up an IDE (select three)

The instructions:
Assume that you are using an Integrated Development Environment (IDE) to develop your next app. Assume that it is as barebones as it gets - any functionality you will have to add yourself. You are now setting up the environment and selecting what plugins to work with to tailor the IDE's functionality to your preference. However, you may only select a maximum of 3 plugins. What selection do you make?

The possible choices to select up to three from:
- ✔ Static code analyzer with security checking rules
- ☐ Bug spotter
- ☐ Syntax highlighter
- ☐ Revision control (CVS)
- ☐ Unnecessary code remover
- ☐ Screen darkener
- ☐ Coding style enforcer
- ☐ UML / Visual modeling

Task 2: Fixing source-code (select one)

The instructions:
Assume that you were given a piece of source-code written by somebody else. They noted having concerns about three parts of the code, and want you to fix them. However, you only have time to address and fix one of these concerns. Given the source code below, please click on the part of the code you would prioritize in fixing.

The possible choices to select from:
- ✔ a security concern where passwords were stored in clear text
- ☐ a performance concern where an inefficient sorting algorithm was used
- ☐ a readability concern where significant hard coded values and lines of code were poorly formatted

Task 3: Seeking help on a confusing API (card sort)

The instructions:
Assume that you are setting up a secure connection to a web server in your app. You need to use a cryptographic API, but have trouble understanding how to use it. Below are a number of approaches you can look for help. Sort these in the order you're most likely to take them.

The choices to sort:
- ✔ Read the API's official documentation
- ○ Read an app development book for your relevant platform
- ○ Ask another developer you happen to know in person
- ○ Search the web, using any resource you may find
- ✘ Go to a specific online resource like StackOverflow to ask for help and use a working example

Task 4: Seeking testers (card sort)

The instructions:
Assume that you have been working on a side-project by yourself, in your own time. You're ready to launch your app, but how do you test it? Sort the testing approaches below in the order you're most likely to take them.

The choices to sort:
- ✔ Send it to a user group for testing
- ○ Ask another developer you happen to know in person to test it
- ○ Ask some direct personal connections (friends, family) to test it
- ○ Test it as well as you can by yourself and then publish it
- ✘ Publish it right away and rely on user feedback for testing

Task 5: Selecting an advertisement SDK (card sort)

The instructions:
Assume you're struggling to get any financial gain from an app you wrote. You decide to incorporate advertisements into the app. There are a couple of approaches you could take. Sort these in the order you're most likely to use them.

The choices to sort:
- ✔ Select specific ad libraries depending on their required permissions
- ○ Select specific ad libraries, depending on how other developers and tech websites rate them
- ○ Select specific ad libraries, purely depending on revenue potential
- ○ Select one or two ad libraries at random and incorporate them
- ✘ Incorporate as many ad libraries as you can

Task 6: Building a software license agreement (card sort)

The instructions:
Assume that you have finished developing a new app. Before releasing it, you talked to a technology lawyer advising you how to best protect you and your new app. He suggested several clauses you should include in the software license for your app, which are given below. Sort these clauses according to how you would prioritize their inclusion into your software license agreement.

The choices to sort:
- ○ A limitation on liability
- ○ A forced arbitration clause
- ○ A provision that allows for disabling any functionality or code by the licensee
- ○ An explicit disclaimer of warranty
- ○ A clause that allows you to terminate the license at any time
- ○ A clause explicitly stating you license the software, not sell it
- ✘ A prohibition on reverse engineering and/or benchmarking

For a task's solution to be considered prioritizing security, the below conditions should be met:

✔ – ***must*** be selected; ✔ – ***must*** be sorted to the top (*prioritized*); ✘ – ***must*** be sorted to the bottom (*de-prioritized*)

Figure 2: Task materials used in Study I. Tasks 3–6 are shown sorted into a security prioritizing order – note that the order of choices presented to participants was randomized in the study. Choices that should be prioritized are highlighted by a green checkmark, whereas choices that should be deprioritized are highlighted by a red cross. Each task was followed by a question to elicit a participant's rationale, phrased as "Please explain why you [chose this option / selected these plugins / sorted the answers in this order]".

(Task 1) whose *order was randomized*. We assessed whether participants prioritizes security by checking if they selected security-supporting options.

Tasks 3–6 were card sorting tasks, where participants were given a number of choices *presented to them in random order* (see Fig. 2) and were asked to sort them in the order they would take them. We assessed whether participants sorted options that supported security as top choice (*prioritizing them*), and sorting options that undermined security as bottom choice (*deprioritizing them*).

Task 1: Setting up an IDE IDEs are commonplace, and can be configured to provide necessary support to developers. The selection was drawn from a list of most popular plugin types. Of the given type of plugins, the static analyzer with security checking rules is the only plugin explicitly mentioning support for automatic analysis of code vulnerabilities, and its inclusion can reasonably be assumed to follow from an attitude of prioritizing security.

Task 2: Fixing source-code Storing passwords in plain text is an obvious security concern which should be avoided. If participants select any other code fragment, they de-prioritize security (or do not have the expertise and/or security-focused approach to development). Thus, to prioritize security, participants should select the security concern. An example of the code presented to participants (one of three randomly shown variations to rule out priming bias) is shown in the online appendix [45].

Task 3: Seeking help on a confusing API Research has shown that copy pasting solutions from StackOverflow does not benefit security [29]. Moreover, in a secure coding experiment, it was shown that between four groups only allowed to use either official API documentation, app development book, web searches, or StackOverflow, it was the group who only used StackOverflow that produced the least secure solutions. In contrast, the group only allowed to use official API documentation produced the most secure solutions [3]. Thus, to prioritize security, participants should first read the API's documentation, and only as a last resort go to StackOverflow to copy-paste an example—and not just trust any example.

Task 4: Seeking testers Research on app testing and release has shown that 'just' releasing apps is fraught with issues [26]. Using dedicated user groups for testing helps to ensure that usability and security do not oppose each other [16]. Thus, app testing via a user group prioritizes a comprehensive testing approach while just publishing the app without any testing de-prioritizes security.

Task 5: Selecting an advertisement SDK There has been a steady increase in the number of permissions used by ad libraries [11]. Research has stressed the need to carefully assess the permissions requested by Android advertisement libraries before incorporating them [44] to avoid accidentally including malicious libraries. Thus, to prioritize security, participants should rank analyzing the required permissions of the ad library highly (vice versa, a higher ranking for including ad libraries at random, or including as many as possible, de-prioritizes security).

Task 6: Building a software license agreement Research has shown that prohibiting reverse engineering and bench marking promotes discovery and sharing of vulnerabilities in black hat communities, while denying this to white hat communities [12, 14]. Thus, to prioritize security, prohibitions on reverse engineering and/or bench marking should not be included.

We trialed an initial study among five professionals with a background in computer science and/or programming. Their feedback was used to verify the estimated time needed to complete the survey (30±5 minutes) and remove any potential misunderstandings in the phrasing and unintentional connotations of demonstrative pictures of developers. The final version of the study was trialed with the same five professionals, after which no further remarks on its structure were found. Before conducting the study, we obtained IRB approval. No personally identifying information was collected in the study itself.

3.1 Participants

We used social networks and public mailing lists to approach app developers, regardless of platform (e.g., Android, iOS). In particular, we solicited participation in the study via relevant professional groups on LinkedIn and Facebook by searching for the terms 'mobile software' or 'app' and 'developer', as well as 'iOS' and 'Android'. After identifying and joining several relevant groups we snowballed for more potential sources of participants by looking at linked groups.

Since our goal was to study the rationale used by app developers across the entirety of app development, we implemented a strict exclusion criterion of any developers who had not actually released/published at least one app on an app store (e.g., Google Play, Apple App Store). While this significantly reduced the potential population, it was necessary in order to establish a realistic picture of developers who can be expected to have had to deal with the different dimensions of app development.

3.2 Materials and procedure

The tasks given to the participants, see Fig. 2, reflect the activities that developers typically undertake [18] (cf. Fig. 1). Each task was followed by an open question eliciting the rationale of their solution. The tasks were presented as part of an online questionnaire that also included background questions: 24 professional and 4 basic demographics (age, gender, nationality, education level). We neither mentioned the security focus of the study nor elicited (self-reported) security experience in the study. This would likely create demand characteristics and, therefore, exaggerate pro-security answers (relative to real-world decisions). Moreover, self-reported security experience would be difficult to interpret in a way that compares objectively across participants as, for example, simply measuring acquaintance with OWASP Top 10 lists does not mean participants have also meaningfully engaged with such knowledge. The study was open for four months, from March to June 2018. We posted reminders in social media groups after two weeks to attempt to elicit additional responses. All raw data of Study I is available online at the online appendix [45].

3.3 Data analysis

Pre-processing. In total, we received 47 complete responses. Before analysis, we manually detected any suspicious entries, discarding three responses based on repetition of nonsensical texts as answers to all open questions. This led to a final set of 44 usable responses.

Task analysis. To assess whether a task's solution indicated a potential security attitude (**RQ1**) we used the options as per Fig. 2:

Task 1 should have at least "static code analyzer with security checking rules" selected.

Task 2 should have "security concern" selected.

Task 3 should have "read the API's official documentation" sorted as 1st or 2nd choice, while "Go to a specific online resource like StackOverflow to ask for help and use a working example" should be sorted as 4th or 5th choice.

Task 4 should have "send it to a user group for testing" sorted as 1st or 2nd choice, while "publish it right away and rely on user feedback for testing" should be sorted as 4th or 5th choice.

Task 5 should have "select specific ad libraries depending on their required permissions" sorted as 1st or 2nd choice, while "incorporate as many ad libraries as you can" should be sorted as 4th or 5th choice.

Task 6 should have "a prohibition on reverse engineering and/or benchmarking" sorted as 5th, 6th, or 7th choice.

Rationale analysis. To assess whether the tasks were performed with a security rationale, and whether this correlated to their solution (**RQ2**), we analyzed the rationale elicited from participants after each task using *open coding*[1]. Two authors independently coded each result to determine whether the rationale indicated a focus on security. Inter-rater reliability measure (Cohen's Kappa [23]) showed strong to perfect inter-rater agreement and reliability across all tasks (ide $k = 0.91$, code $k = 0.91$, help $k = 1$, testing $k = 0.85$, ads $k = 0.87$, legal $k = 1$).

Thematic analysis. We used thematic analysis [13] to further understand participants' rationales (**RQs 1–3**), especially those that were not security related. Two authors independently coded the rationale data and compared codebooks, agreeing on dominant themes emerging from the analysis.

Statistical tests. Task solutions and rationales were binarily classified (secure, other). Binominal tests are most appropriate for this type of categorical data. We, therefore, used binomial tests to assess whether rationales were more or less likely to appear than by chance, and whether these distributions differ between tasks.

4 STUDY I – FINDINGS

4.1 Demographics – Who is the archetypal app developer?

The range of mobile developers we found in Study I is in line with existing research [17, 42] and may effectively be stereotyped as young independent men with at least a college degree, building apps to improve their skills and know-how and have fun, while being heavily reliant on knowledge from community websites like StackOverflow.

Demographics. Participants were predominantly male (89%), some female (9%) and one other (2%). Geographically, they are distributed primarily across North America (32%), Europe (39%), and the Middle East (27%), with one participant from Asia (2%). The majority were young, with some published app developers even being under age, < 18 (5%), 18–24 (18%), 25–34 (41%), 35–44 (23%), 45–54 (11%), 55–64 (2%). Most finished a tertiary education, with highest completed education being vocational training (9%), bachelor (55%), master (20%), doctoral (9%).

The professional context. Most developers in the study are fairly junior, having released (or been involved in) one to five apps (68%), six to ten (25%), and only a few with 11+ (7%). The biggest sub-group was solo developers (36%), followed by part of micro-enterprises with fewer than 10 people (23%), and finally SMEs (20%) and large enterprises (20%). Similarly, development team size tended toward smaller teams: solo developers (39%), two to five people (39%), teams of six to nine people (11%), 10+ people (11%). Even with most participants having finished a tertiary qualification, the majority professed to be self-taught for app development (80%), many also noting contributions from their education (59%), and a much smaller group mentioning training at work (20%).

The development focus. Participants developed mostly for Android (77%), with iOS (55%), and several others including Windows Phone (7%). There is a fairly even split between those who develop exclusively for one platform (57%), and those who develop for multiple platforms (43%). Of those developing for multiple platforms, 42% released simultaneously for all platforms (typically Android and iOS). Most developed to improve their skills and know-how (68%), to have fun (66%), or for intellectual stimulation (55%). Somewhat less-represented motivations include no apps with similar functionality existing yet (41%), financial gain (39%), and finally, building their reputation within the app developer community (34%).

4.2 Task results – Do solutions indicate a prioritization of security?

Figure 3 presents the results of the tasks described in Sec. 3.3. Fig. 3(a) shows the distribution of solutions, namely those that were deemed to prioritize security vs. those that did not while Fig. 3(b) illustrates the distribution of the various rationales. Detailed task results are shown in the online appendix [45].

Roughly half of the participants provided a solution prioritizing security for tasks 1, 2, 5, and 6 (respectively 50%, 48%, 52%, and 52%), although those doing so with explicit security rationales were far fewer. Tasks 3 and 4, in which participants had to reason about (the need for) social interaction with, and reliance upon others (i.e., asking them to test or overcome a coding challenge), showed fewer solutions prioritizing security (23% and 14%). In other words: participants rarely offered a security rationale for tasks not involving code, regardless of whether their solution prioritized security.

Secure rationales are asymmetrical (differing from chance, $p<0.01$) except for task 2 where the distribution does not differ from chance. Binomial test found that the proportion of secure rationalization differs significantly for all non-coding tasks when compared to coding tasks ($p<0.01$), suggesting that secure rationalizations are less likely to appear for non-coding tasks. Binomial tests showed that tasks 3 and 4 differ significantly from chance levels ($p<0.05$, $p<0.01$, two tailed) whereas for task 1, 2, 5, and 6 these frequencies of prioritization do not differ from a distribution that would arise by chance ($p>0.88$).

4.3 Rationale analysis – What is in the Box?

We performed two distinct thematic analyses to build a holistic understanding of developer rationales: one to understand the rationales driving the choices made in each task, and another to understand cross-task themes. For the first analysis, key themes prevalent in the codebook are shown in Fig. 4. The three subgroups most relevant to understand why developers do (not) develop securely are discussed here: (1) security rationales and security solutions, (2) security rationales but non-security solutions, and (3) non-security rationales with security solutions. Distribution of these subgroups over the professional demographics of the participants showed no significant deviation across organization size.

4.3.1 Security rationales, and security solutions.

Trust in other developers: Across different tasks participants

[1] As in qualitative analysis [15] and not to be confused with *code* in a software sense.

noted that they would take into account other developers' feedback and opinions when making their own decisions about fixing source-code:

> "Another developer is adequately smart to give me technical feedback" (P13)

Or while deciding which ad library to incorporate, the knowledge and opinion of other developers were considered a valuable resource:

> "[…] I will use the library(ies) that is most used to gather the fastest support either from developers and community." (P13)

This may further explain the quantitative finding that developers rationalize about security less in tasks involving social interactions with other developers because of the trust they place in them.

Understanding app permissions: While participants showed trust in other developers when deciding on permissions to incorporate within their app:

> "see what others have done and then look at permissions and revenue potential." (P20)

they also showed an awareness of the importance of permissions and the need to carefully assess which permissions to request (and the value they would derive in return):

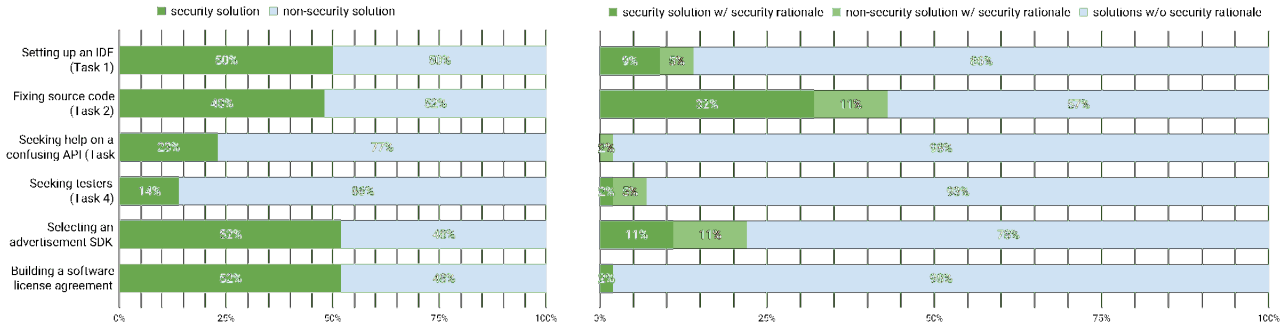
> "Permissions required are important and I like to match it to my apps to ensure a good return [on investment]." (P41)

Don't implement your own crypto: Participants showed a clear understanding of the risks of hand-rolled cryptography code:

> "While the crypto looks iffy (why aren't we just calling a library hashing function), […]" (P24) "The hashing algorithm is a hand-rolled version of SHA-256 which might be buggy, […] (P38)

and strongly recommended changing hand-rolled to library use:

> "I would tell them not to store plaintext passwords of their users, and to use a secure hash function from a library (using a seed together with the password) rather than implementing their own." (P19)



(a) Distribution of solutions

(b) Distribution of solutions with and without and secure rationale

Figure 3: Distribution of (a) task solutions according to Fig 2's classification, and (b) task rationale.

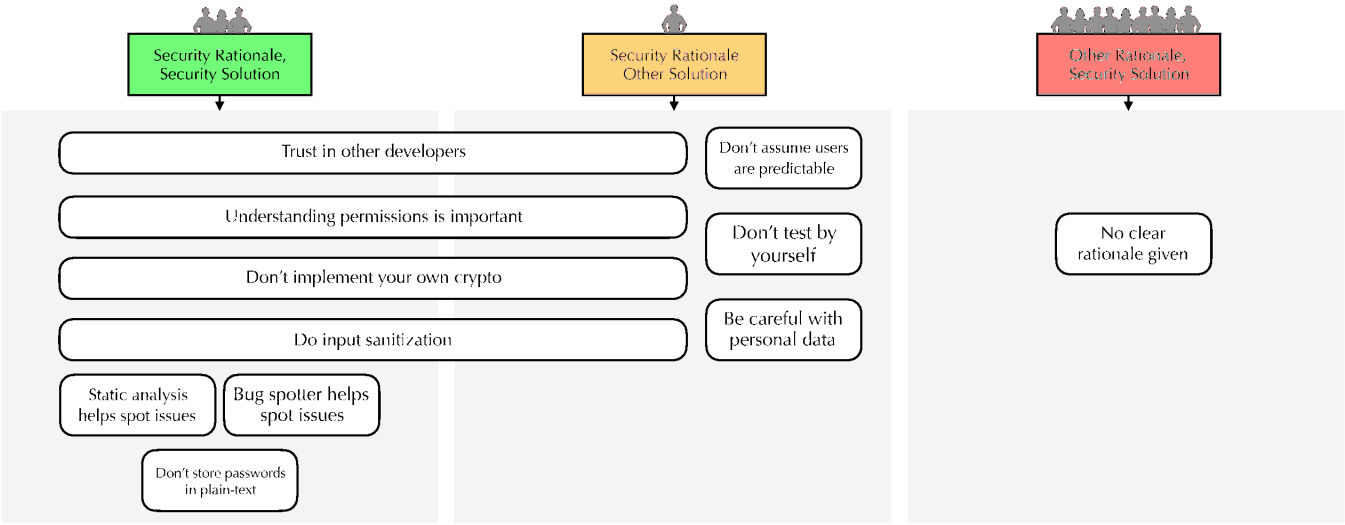


Figure 4: Schematic depiction of the codebook. Thematic analysis over the rationales for the tasks divided into three most relevant subgroups: (a) security rationales with security solutions, (b) security rationales, but with non-security solutions, and (c) Other rationales but with security solutions.

This indicates that messaging about *not rolling one's own crypto* is reaching app developers.

Do input sanitization: Several participants noted this, whether by explicitly pointing out that the source-code in task 2 lacked this:

"Variables haven't been sanitised." (P39)

to more nuanced statements explaining one should not just trust data provided by others:

"…just inserting untrusted, unsanitised data is a bad idea" (P32)

Don't store passwords in plain text: Participants showed an understanding of the risks of storing passwords in plain text, whether simply calling it out as being not done:

"Passwords should never be stored in plain text." (P39)

to re-iterating *why* this should not be done:

"This stores the user's plaintext password in a file. If the file was compromised then user's passwords would be leaked. Passwords should never be stored – Only the hash." (P28)

Static analysis and bug spotters help spot issues: Participants also showed a clear understanding of their own limitations in catching any and all possible coding mistakes or vulnerabilities, some noting additionally how important this is when developing alone without the safety net of a wider team:

"Bug spotter and analyzer because it'd help fix issues I create which is important as a lone dev with no code review" (P9)

"The most important thing would be detecting things that are 'off' in the code. Syntax highlighting helps me do this myself, and the bug spotter / static code analyzer plugins do this in an automated fashion." (P19)

Moreover, participants noted the importance of such tools because even when armed with the relevant security knowledge, they

"[…] help prevent non-obvious security issues." (P24)

4.3.2 Secure rationales, but other solutions. This subgroup, smaller in number than others, shared a number of themes with those who chose security solutions and provided security rationales, namely, trusting in other developers, and more technical matters like understanding permissions, not implementing their own crypto, and always performing input sanitization.

Be careful with personal data: The one particular theme that came to the fore within this subgroup is that participants always considered whether they would have to process and hold personal data of their users, and adjusted their decision-making accordingly:

"I'm assuming that if I'm writing an app that requires the user's personal information I would first make sure that I am well-versed in the field of data security." (P6)

Some participants noted the same reservations in the context of specific resources such as ad libraries:

"I would want to know what data libraries want to collect on my users and what they plan on doing with it." (P24)

Do not test by yourself and don't assume users are predictable: This again varied from participants simply calling it out,

"A developer should never test the code he built." (P1)

to giving a more detailed reasoning, e.g., that one can never predict how software is actually used, and thus account for this as much as possible:

"I never rely on just myself for testing, since I cannot always know how people are going to use the app beforehand." (P1)

Similarly, developers gave more technical considerations to dealing with the unpredictable nature of their users:

"Never trust user input. Sanitize it before doing anything …" (P18)

4.3.3 Other rationales, but security solutions. By far the largest subgroup in the analysis, many participants produced a secure solution, but seemingly almost by accident. Unclear rationales such as "no specific reason" (P18), or simply because it's the "best [solution] I could come up with" (P34), or that solutions "looked OK like this." (P14) means that we are unable to determine their intentions (i.e. focusing on security or not).

Through some latent knowledge or behavior, participants produced security relevant solutions – yet we cannot be sure why they solved the tasks the way they did. From a security standpoint this is particularly troubling, as there is no insight into whether this group of developers are consistently acting in a secure way (but subconsciously), following secure development trends, or simply doing whatever feels right at the moment.

4.3.4 What else characterizes participants' rationales? The thematic analysis of the rationale data gave rise to several recurring themes that spanned across different tasks. These are described below to allow for further understanding of why developers may prioritize the way they do.

Caring about users: Many participants, across several tasks, reflect on the impact of their decisions on users of the app. This manifests in *altruistic* or *empathetic* context as, for example, in reflection on the advertisement and license tasks:

"I don't want to put my users at risk with weird permissions." (P30)

"I prioritized based on how 'extortionist' or 'evil' the clauses would feel to me as a consumer." (P19)

Additionally, participants manifest the attitude in an *egotistic* context as well. For example, caring about users in order to preserve or improve one's reputation as a developer:

"[…] it is better to have a small group [of testers] find bugs than your users. This way, your reputation would be secured with the whole world." (P3)

Avoiding dealing with others: In the tasks where participants had to reason about interacting with others, a theme arose indicating their reluctance to engage with others before putting in their own effort. For example:

"This problem should be solved as fast as possible but *without being needy* so firstly I would read documentation and try to find a solution on my own." (P17) (emphasis added)

"First I'll use resources I can use without the help from other people (their time is important as well). (P7)

All about the flow: In rationales for setting up an IDE, a key distinction was IDE configurations supporting the developer getting into a state of flow during development, rather than selecting plugins that intervene and steer their behavior:

"Moving forward and getting better at programming takes practice and I really enjoy the process. Programming at my own speed without interruptions from plugins is my most productive workflow." (P30)

Similar examples manifested in the testing task, stressing the importance of flow:

> "This is the easiest way to move forward in the process without having to get out of my chair. My time at work is limited. (P30)

An easy does it attitude: In the coding task, a distinction arose between developers who take a pragmatic approach to fixing, and those who focus on key issues arguably important to them. Many participants take an approach of dealing with the part perceived as being easiest. For example, P2, who provided a solution prioritizing security in the coding task, did so for rather pragmatic reasons:

> "The least complex block. Other blocks either have too many notes and documentation to read and understand." (P2)

This theme manifested across other tasks, showing pragmatic behaviors rooted in familiarity, such as justification of how to test:

> "Launching an app is really not a big deal. A soft launch is often best and you shouldn't stress out about it. However, may as well test some. :)" (P36)

Moreover, in the context of asking for help, some participants very explicitly showed an easy does it attitude justifying their behavior:

> "Least to most effort, I'm very lazy you see" (P9)

A misunderstanding of licensing: Many participants, even though being solo developers with limited to no resources to afford legal advice, have little understanding or patience for what license clauses mean, or how they affect themselves and their customers. This manifested mostly as participants flat out rejected the importance of licensing and the effects a Software License Agreement (SLA) has on their products (let alone themselves):

> "Sorry, I don't see how it matters" (P43)

Several other participants were guided by ideological stance, in these cases support of Open Source movements, underestimating the role that an SLA plays, and the fundamental distinction between license and product:

> "I don't see the need for licensing. I am a large proponent of FOSS and the 'it's not my fault if your house burns down' clause" (P9)
> "Licensing is irrelevant, because the app is free" (P25)

5 STUDY II – PERCEIVED SECURITY IMPACT OF DEVELOPMENT ACTIVITIES

To help generalize findings from Study I, we adapted its tasks into a survey with Likert scale questions, eliciting the perceived impact of the tasks on security. The survey was approved by the IRB.

5.1 Participants

We collected a random sample of 60,163 email addresses of Android application developers listed in Google Play. We emailed these developers requesting their participation in the survey. We offered no reward, and noted that participation was voluntary. After sending out the invitations, 3419 emails bounced; we honored another 3375 requests to not be contacted, leaving 53,639 emails sent. A total of 291 completed the survey over the two weeks period for which the study was active. From the 291 responses, we discarded two partial responses and additionally discarded fourteen potentially suspicious responses which scored all items the same. This led to a final set of 274 usable responses, well in line with other security research employing this recruitment strategy (cf. [2, 3]).

5.2 Materials and Procedure

We used a single survey with six questions aligned with Study I's tasks. Participants were asked to rate from strongly disagree (-2) to strongly agree (+2) on whether:
The security of the app I develop is affected by decisions I make on …

- …the code I write (*code*)
- …setting up my development environment (*dev*)
- …the license my software is released under (*license*)
- …the external SDKs I use (*sdk*)
- …whom I involve in testing (*test*)
- …what sources I go to for help (*help*)

Appropriate for the ordinal type data elicited by the above Likert scale questions, we used polychoric correlation [34] to estimate the correlation (r) between the answers. All raw data of Study II is available online at the online appendix [45].

6 STUDY II – FINDINGS

6.1 Questionnaire Results

Figure 5 illustrates the results from the Likert scale questions – whether different activities are perceived to affect the security of software developed. Participants rated that the decisions they made while writing code and using external SDKs affect the security of their software, while the license under which their app is released was perceived to not affect security. Other tasks like "whom I involve in testing", "sources I got to for help" etc. show a more balanced distribution, indicating there is no clear consensus on whether decisions made in such tasks affect the security of an app. Items skewing towards positive responses may indicate aspects where developers are more likely to consider security.

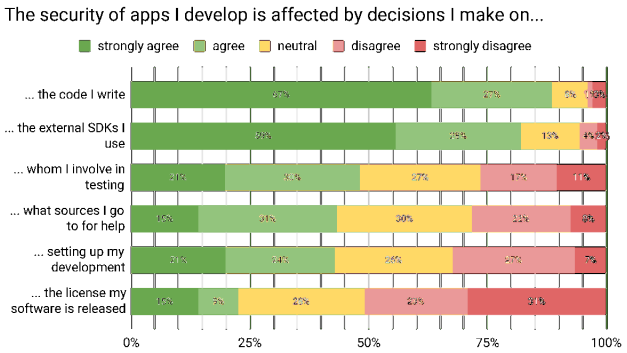


Figure 5: App developers' perception (N=274) whether development activities affect the security of their app.

6.2 Questionnaire Analysis

To understand whether developers' attitudes may be linked between similar activities, we calculated correlations between the different activities (polychoric r being more appropriate for ordinal value assessing rater/developer agreement) In line with link between writing code and using external SDKs found above, there was a positive (albeit low) correlation between the perceived security

impact of writing code and selecting external SDKs (r=.27, p<0.01), as well as a low negative correlation between writing code and under what license their software is released (r=−.14, p<0.05). This may indicate that writing code and selecting external SDKs are indeed the two key factors which developers perceive to affect the security of their software. Additional correlations were found, in particular the two strongest correlations between the test and help question (r=.39, p<0.01), and the help and development environment question (r=.37, p<0.01).

The correlation between test and help – and that it did not score as affecting security – corroborates the findings from Study I where participants also prioritized security the least in the same tasks.

The correlation between how participants set up their development environment and what sources they go to for help may reflect some ways of working we found in Study I. For example, participants who deem it important to set up their IDE carefully, e.g., by incorporating static analysis and bug spotters, may do so *because* they trust in authoritative sources and communities which set out best practices for secure coding and thereby ensure the security of their apps. This needs to be validated through further research.

7 THREATS TO VALIDITY

Construct Validity. We based the items in Studies I and II directly on activities found in recent literature summarizing the wide spectrum of activities in software development [18]. Moreover, for Study I we took care to ground the definitions of security solutions in relevant literature showing the effect of insecure decisions in licensing [12, 14], incorporation of advertisement SDKs [11, 44], seeking out of testers [16, 26], and use of different sources for help [3, 29]. We do not claim to exhaust the security challenges faced by developers through these tasks, but rather, sample a diverse variety of challenges they must face.

Internal Validity. We do not claim to infer actual behavior from the self-reported data elicited in Studies I and II. Moreover, in Study I we explicitly only investigated estimated correlation between solutions prioritizing security and *any* kind of rationale indicating security. We do not claim that this lack of secure action and rationale implies a lack of knowledge, which requires further psychological work.

Moreover, given the nature of the tasks and their stand-alone design, it necessarily means that participants' decision-making is done in a hypothetical context. This may affect the way in which participants prioritize particular answers over others as, for example, time pressure from managing concurrent tasks and activities in a real-world scenario could lead to different prioritization. This was, however, a necessary compromise in order to reveal the heterogeneity of reasoning that developers hold for different kinds of tasks.

Notably, therefore, the thematic analysis of participants' rationales did not expose any evidence that: they considered real-world contexts differently; their decisions were driven by prior security knowledge; or some choices were taken to be much more likely or 'obvious' than others.

External & Ecological Validity. We used purposive sampling complemented with snowballing for both Studies I and II – specifically wanting to include only those with experience of the variety of activities encountered in app development. While this proved no problem for Study II due to the minimum time investment required of participants, we couldn't recruit as many developers willing to invest the required average half hour into Study I. Nonetheless, a comparison of the personal and professional demographics detailed in Sec. 4.1 indicates significant overlap with demographics established in earlier research [42]. Given that Study I's findings among a much wider sample of the mobile developer population is in line with our key findings of Study II: a security focus on code-related tasks and under-appreciation of security impact of other activities, we feel confident that the samples of both studies show a valid picture of the typical developers encountered in mobile app development.

Moreover, the lack of security knowledge of participants in Study I may limit the generalization of the findings, as we cannot confirm whether security knowledge is normally distributed across participants, or skewed towards a particular extreme. However, eliciting self-reported experience has its limitations as discussed before, and would not necessarily enrich the findings. Thus, further studies may attempt to use objective measures of security knowledge (rather than self-reported experience) in focused task studies to assess whether the level of security knowledge has an effect on different tasks. This was not deemed feasible in the present study due to the need to incorporate a wider variety of tasks, which already led to a study with significant duration, limiting the potential of recruiting willing participants.

8 DISCUSSION: FILLING THE BOX WITH SECURITY

As we mentioned in Sec. 2.3, we need to understand developers' rationales in order to begin conceptualizing interventions to mitigate vulnerabilities stemming from these rationales. Without a holistic focus on app development–looking beyond solely the act of writing code–we cannot effectively support developers in *producing* secure apps, rather than 'just' *writing* secure code. We discuss some key insights leading to takeaways on how app developers can be supported to do so.

8.1 Be social, but also be critical

Studies I and II both indicated that developers lack a critical attitude towards security in activities that involve (the need for) social interaction with others. In particular, the *seeking help* and *seeking out testers* tasks of Study I which required participants to consider third parties and whether they can be trusted showed the lowest amount of security rationalization. This may indicate that developers do not make the leap of thinking of people and sources they use to aid in their app's development as potentially misguided, or even adversarial.

These kinds of social interactions are vitally important for independent and small scale developers, as they need to reach out to external support networks, rather than rely on existing organizational structure. In reality, this will often come down to considering whether libraries and friendly advice or useful code fragments posted on forums such as StackOverflow can be inherently trusted to be provided in good faith, let alone competence. To avoid the risk of incorporating vulnerable code, promoting a more critical attitude

to how social interactions can affect the security apps would be warranted.

Takeaway—to ensure a more critical attitude towards security among developers, we need to promote critical reflection on the trusting of people and resources on which they rely.

8.2 Flowing towards a secure environment

Most of the rationales in task 1 (Study I) – in which participants had to reason about setting up their development environment – focused on setting up the environment enabling them to work in a particular way. So it should not be a surprise that the solutions expressing security considerations would mention so similarly. For example, P9 made their selection because "it'd help fix issues I create which is important as a lone dev with no code review" (P9), and P24 did so because the "static analyzer helps prevent non-obvious security issues." This can be interpreted as the basis for developers getting into a state of 'flow' – they prepare to work in a way where they can be entirely absorbed in that work [32]. The key link between the flow rationale and the security mindset seems to be a realization of those participants who work as solo developers that they cannot expect themselves to catch everything, but that they are still responsible for everything at the end of the day.

Takeaway—solo and small-scale app developers need support to make the 'right' choices in terms of security, but not only for code. Research could focus on incorporating support for such activities within IDEs but in a manner that is unobtrusive to the fluidity that developers desire.

8.3 Thinking about users, thinking about security

Some participants' rationale explicitly showed that they were anticipating the way their users would behave: "I never rely on just myself for testing, since I cannot always know how people are going to use the app beforehand" (P1). Others mentioned considering whom to involve based on their relationships, and how it might then affect their users: "Friends and family are terrible testers, because they don't want to hurt your feelings. You yourself are even worse, and relying on your users will find you all the bugs, but might lose you your reputation" (P20). This hints that explicit consideration of how users are affected by the software they develop leads them to more carefully consider the decisions they make. We find further evidence of this especially in task 5, in which participants had to seek help on incorporating an advertisement library. Here considerations of the users came through in terms of the potential impact of vulnerabilities on users, for example, (P19) noting that they "[...] would not want my users to experience any negative side effects from the ad library," and (P13) mentioning that "it is important to grant minimal privacy impact to users".

Takeaway—to reason about security, and to act upon it, developers should be encouraged to take a user-centric view – understanding the consequences vulnerabilities have for their users.

9 CONCLUSION AND FUTURE WORK

We investigated the rationale of app developers during different activities in the app development process, performing an in-depth task-based study with 44 app developers assessing the extent to which they prioritize security and how they rationalize their choices. We followed this up with a larger survey of 274 app developers assessing their perception towards the impact of the different activities on the security of their produced apps.

Our analysis highlights that app developers rarely prioritize secure ways of working for reasons of security unless it directly affects the writing or the functionality of their code. In *social* (seeking help or testers), *economic* (monetization) and *legal* (licensing) aspects of app development, there were no prevalent security rationales underpinning their choices.

Future work could consider, e.g., developers' relationships with others – e.g., thinking about users and the consequences vulnerabilities have for them or identifying with a wider developer community with its security norms, values and practices – may impact developers' security behaviours and may be leveraged to bring a stronger security rationale across the variety of activities in which they engage during app development.

Acknowledgments

This work is partially supported by EPSRC grant EP/P011799/1, Why Johnny doesn't write secure software? Secure software development by the masses and SFI grant 13/RC/2094.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. 2016. Sok: Lessons learned from android security research for appified software platforms. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 433–451.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 154–171.

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 289–305.

[4] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L Mazurek, and Sascha Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *Cybersecurity Development (SecDev), 2017 IEEE*. IEEE, 22–26.

[5] Hala Assal and Sonia Chiasson. 2018. Motivations and Amotivations for Software Security – Preliminary Results. In *Workshop on Security Information Workers (WSIW) 2018*.

[6] Hala Assal and Sonia Chiasson. 2018. Security in the Software Development Lifecycle. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 281–296. https://www.usenix.org/conference/soups2018/presentation/assal

[7] Hala Assal and Sonia Chiasson. 2019. "Think secure from the beginning": A Survey with Software Developers. In *Proc. CHI'19*.

[8] Andrea Atzeni et al. 2011. Here's Johnny: a methodology for developing attacker personas. In *Proc. ARES'11*. IEEE, 722–727.

[9] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 356–367.

[10] Adam Bauer and Bauer Hebeisen. 2017. Igexin advertising network put user privacy at risk. https://blog.lookout.com/igexin-malicious-sdk. (2017). Online; accessed 20 October 2018.

[11] Theodore Book, Adam Pridgen, and Dan S Wallach. 2013. Longitudinal analysis of android ad library permissions. *arXiv preprint arXiv:1303.0857* (2013).

[12] Jean Braucher. 2006. 12 Principles for Fair Commerce in Mass-Market Software and Other Digital Products. In *Arizona Legal Studies Discussion Paper No 06-05*. Available at SSRN: https://ssrn.com/abstract=730907.

[13] Virginia Braun, Victoria Clarke, and Gareth Terry. 2014. Thematic analysis. *Qual Res Clin Health Psychol* 24 (2014), 95–114.

[14] Jennifer Chandler. 2009. Information Security and Contracts. Contracting Insecurity: Software License Terms That Undermine Information Security. In *Harboring Data: Information Security, Law, and the Corporation*, Andrea M. Matwyshyn (Ed.). Stanford University Press, Stanford, California.

[15] Juliet M Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21.

[16] Lorrie Cranor and Simson Garfinkel. 2005. *Security and Usability*. O'Reilly Media, Inc.

[17] Amy Cravens. 2012. A demographic and business model analysis of today's app developer. *GigaOM Pro* (2012).

[18] Cleidson RB de Souza et al. 2016. The social side of software platform ecosystems. In *Proc. CHI'16*. ACM, 3204–3214.

[19] William Enck et al. 2011. A Study of Android Application Security.. In *Proc. SEC'11@USENIX*, Vol. 2. 2.

[20] Ecular Xu (Mobile Threat Response Engineer). 2017. Analyzing Xavier: An Information-Stealing Ad Library on Android. https://blog.trendmicro.com/trendlabs-security-intelligence/analyzing-xavier-information-stealing-ad-library-android/. (2017). Online; accessed 20 October 2018.

[21] Sascha Fahl et al. 2012. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proc. CCS'12*. ACM, 50–61.

[22] Felix Fischer et al. 2017. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *Symp. S&P'17*. IEEE, 121–136.

[23] Kevin A Hallgren. 2012. Computing inter-rater reliability for observational data: an overview and tutorial. *Tutorials in quantitative methods for psychology* 8, 1 (2012), 23.

[24] Julie M Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. 2018. "We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS) 2018*. 357–373.

[25] Ankur Kataria, Tricha Anjali, and Raghu Venkat. 2014. Quantifying smartphone vulnerabilities. In *Signal Processing and Integrated Networks (SPIN), 2014 International Conference on*. IEEE, 645–649.

[26] Eero Laukkanen, Maria Paasivaara, Juha Itkonen, and Casper Lassenius. 2018. Comparison of release engineering practices in a large mature company and a startup. *Empirical Software Engineering* (2018), 1–43.

[27] Tamara Lopez, Thein Tun, Arosha Bandara, Bashar Nuseibeh, Helen Sharp, and Mark Levine. 2018. An Investigation of Security Conversations in Stack Overflow: Perceptions of Security and Community Involvement. In *First International Workshop on Security Awareness from Design to Deployment (SEADâĂŹ18)*.

[28] Manuel Maarek, Sandy Louchart, Léon McGregor, and Ross McMenemy. 2018. Co-created Design of a Serious Game Investigation into Developer-Centred Security. In *International Conference on Games and Learning Alliance*. Springer, 221–231.

[29] Na Meng, Stefan Nagy, Danfeng Yao, Wenjie Zhuang, and Gustavo Arango-Argoty. 2018. Secure coding practices in java: Challenges and vulnerabilities. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 372–383.

[30] Ibtisam Mohamed and Dhiren Patel. 2015. Android vs iOS security: A comparative study. In *Information Technology-New Generations (ITNG), 2015 12th International Conference on*. IEEE, 725–730.

[31] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why do developers get password storage wrong?: A qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 311–328.

[32] Jeanne Nakamura and Mihaly Csikszentmihalyi. 2014. The concept of flow. In *Flow and the foundations of positive psychology*. Springer, 239–263.

[33] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. API Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, 315–328. https://www.usenix.org/conference/soups2018/presentation/oliveira

[34] Ulf Olsson. 1979. Maximum likelihood estimation of the polychoric correlation coefficient. *Psychometrika* 44, 4 (1979), 443–460.

[35] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2014. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications.. In *NDSS*, Vol. 14. 23–26.

[36] Venkatesh-Prasad Ranganath and Joydeep Mitra. 2018. Are Free Android App Security Analysis Tools Effective in Detecting Known Vulnerabilities? *arXiv preprint arXiv:1806.09059* (2018).

[37] Bradley Reaves, Jasmine Bowers, Sigmund Albert Gorski III, Olabode Anise, Rahul Bobhate, Raymond Cho, Hiranava Das, Sharique Hussain, Hamza Karachiwala, Nolen Scaife, et al. 2016. * droid: Assessment and evaluation of Android application analysis tools. *ACM Computing Surveys (CSUR)* 49, 3 (2016), 55.

[38] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.

[39] Joe Rossignol. 2015. What You Need to Know About iOS Malware XcodeGhost. https://www.macrumors.com/2015/09/20/xcodeghost-chinese-malware-faq/. (2015). Online; accessed 20 October 2018.

[40] Alireza Sadeghi, Hamid Bagheri, Joshua Garcia, and Sam Malek. 2017. A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. *IEEE Transactions on Software Engineering* 43, 6 (2017), 492–530.

[41] David Scott and Richard Sharp. 2002. Developing secure web applications. *IEEE Internet Computing* 6, 6 (2002), 38–45.

[42] SlashData Developer Economics. 2016. Developer Economics: State of the Developer Nation Q1 2016. https://www.developereconomics.com/reports/developer-economics-state-of-developer-nation-q1-2016. (2016). Online; accessed 18 September 2017.

[43] SlashData Developer Economics. 2017. Developer Economics: State of the Developer Nation. https://www.developereconomics.com/reports. (2017). Online; accessed 20 October 2017.

[44] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. 2012. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, Vol. 10.

[45] van der Linden, Dirk and others. 2020. Schrödinger's Security (ICSE 2020) Appendices. http://hdl.handle.net/1983/f43803de-4ade-488f-be1a-a2e8ba30c201. (2020). Online; accessed 8 January 2020.

[46] Daniel Votipka, Michelle L Mazurek, Hongyi Hu, and Bryan Eastes. 2018. Toward a Field Study on the Impact of Hacking Competitions on Secure Development. In *Workshop on Security Information Workers (WSIW)*.

[47] Charles Weir et al. 2016. How should mobile app programmers learn security? Comparing and contrasting expert views. In *Proc. SOUPS@USENIX'16*.

[48] Chamila Wijayarathna and Nalin Asanka Gamagedara Arachchilage. 2018. Am I Responsible for End-User's Security? A Programmer's Perspective. *arXiv preprint arXiv:1808.01481* (2018).

[49] J. Xie, H. R. Lipford, and B. Chu. 2011. Why do programmers make security errors?. In *Symp. VL/HCC 2011*. IEEE, 161–164.