You Get Where You're Looking For
The Impact of Information Sources on Code Security

Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim†, Michelle L. Mazurek†, Christian Stransky
CISPA, Saarland University; †University of Maryland, College Park

***Abstract*—Vulnerabilities in Android code – including but not limited to insecure data storage, unprotected inter-component communication, broken TLS implementations, and violations of least privilege – have enabled real-world privacy leaks and motivated research cataloguing their prevalence and impact. Researchers have speculated that appification promotes security problems, as it increasingly allows inexperienced laymen to develop complex and sensitive apps. Anecdotally, Internet resources such as Stack Overflow are blamed for promoting insecure solutions that are naively copy-pasted by inexperienced developers.**

In this paper, we for the first time systematically analyzed how the use of information resources impacts code security. We first surveyed 295 app developers who have published in the Google Play market concerning how they use resources to solve security-related problems. Based on the survey results, we conducted a lab study with 54 Android developers (students and professionals), in which participants wrote security- and privacy-relevant code under time constraints. The participants were assigned to one of four conditions: free choice of resources, Stack Overflow only, official Android documentation only, or books only. Those participants who were allowed to use only Stack Overflow produced significantly less secure code than those using, the official Android documentation or books, while participants using the official Android documentation produced significantly less functional code than those using Stack Overflow.

To assess the quality of Stack Overflow as a resource, we surveyed the 139 threads our participants accessed during the study, finding that only 25% of them were helpful in solving the assigned tasks and only 17% of them contained secure code snippets. In order to obtain ground truth concerning the prevalence of the secure and insecure code our participants wrote in the lab study, we statically analyzed a random sample of 200,000 apps from Google Play, finding that 93.6% of the apps used at least one of the API calls our participants used during our study. We also found that many of the security errors made by our participants also appear in the wild, possibly also originating in the use of Stack Overflow to solve programming problems. Taken together, our results confirm that API documentation is secure but hard to use, while informal documentation such as Stack Overflow is more accessible but often leads to insecurity. Given time constraints and economic pressures, we can expect that Android developers will continue to choose those resources that are easiest to use; therefore, our results firmly establish the need for secure-but-usable documentation.

I. Introduction

Mobile devices in general and Android in particular are a rapidly growing market. Globally, mobile digital media has recently surpassed desktop and other media [37]; billions of users and devices with millions of apps installed attract many (new) developers. Previous research has found that many of these mobile apps have poorly implemented security mechanisms, potentially because developers are inexperienced, distracted or overwhelmed [1], [8], [9], [11], [14]–[18], [26], [29], [31], [33], [34], [36], [43], [44], [46]. Developers tend to request more permissions than actually needed, do not use TLS or cryptographic APIs correctly, often use insecure options for Inter Component Communication (ICC), and fail to store sensitive information in private areas.

Some previous work attempts to assess root causes for these programming errors. A frequent conclusion is that APIs are too complicated or insufficiently documented. Anecdotal reports indicate that developers use a search engine for help when they encounter an unfamiliar security issue. The search results often lead to official API documentation, blog posts, or Q&A forums such as Stack Overflow[1]. For example, Fahl et al. [16]–[18] interviewed developers whose use of pasted code snippets from Stack Overflow made them vulnerable to Man-In-The-Middle attacks.

These anecdotes set the stage for our work: While many developer issues have been identified in recent years, we know only very little about how these security issues make their way into apps, and most of what we know remains unsubstantiated. In this paper, we assess the validity of these anecdotes by exploring the following research questions:

- What do Android developers do when they encounter a security- or privacy-relevant issue?
- Which information sources do they use to look up security- or privacy-relevant questions?
- Does the use of Stack Overflow really lead to less secure code than the use of other resources?
- Is the official Android documentation really less usable, resulting in less functional code compared to other resources?

We are the first to address these questions systematically rather than anecdotally, shedding light on the root causes of security-related programming errors in Android apps. In order to understand these causes, we first conducted an online survey of 295 developers with apps listed in the Google Play marketplace, covering how they handle both general and security-specific programming challenges in their daily work. We found that most developers indeed use search engines and Stack Overflow to address security-related issues, with a sizable number also consulting the official API documentation and a few using books.

Based on the results of this study, we conducted a laboratory user study with 54 student and professional Android developers, assessing how they handle security challenges when given different resources for assistance. Participants were assigned to one of four study groups, in which we isolated *conditions*:

[1] http://stackoverflow.com

free choice of resources, Stack Overflow only, official Android documentation only, and books only. Each participant was asked to complete four programming tasks that were drawn from common errors identified in previous work: A secure networking task, a secure storage task, an ICC task, and a least permissions task. We analyzed the correctness and security of the participants' code for each task as well as how they employ the resources we permitted them to use. Our results validate the prior anecdotal evidence: Participants using Stack Overflow were more likely to be functionally correct, but less likely to come up with a secure solution than participants in other study conditions.

To place these results in context, we also surveyed the quality of Stack Overflow Q&As. We first analyze the relevance and security implications of the 139 Stack Overflow threads accessed by our subjects. We found that many of the threads contain insecure code snippets, and that those threads are equally as popular as threads with only secure snippets.

To establish ground truth, we also apply static analysis to a random sample of 200,000 free apps from the Google Play market in order to investigate if the code written in the context of our laboratory study can be found in the wild. We find that our programming tasks were highly representative for the typical Android programmer, as 93.6% of all apps we analyzed used at least one of the API calls our participants generated during the study. Our analysis also finds that many of the security errors made by our participants when using these APIs also appear in the wild. For example, most of the custom hostname verifier implementations we found in real-world apps implement insecure hostname verification, which is also true for the code written by our participants.

Taken together, our results confirm an important problem: Official API documentation is secure but hard to use, while informal documentation such as Stack Overflow is more accessible but often leads to insecurity. Interestingly, books (the only paid resource) perform well both for security and functionality. However, they are rarely used (in our study, one free choice participant used a book). Given time constraints and economic pressures, we can expect that Android developers will continue to choose those resources that appear easiest to use; therefore, our results firmly establish the need for secure-but-usable documentation.

The rest of this paper proceeds as follows: In Section II we review related work. Section III describes our online survey of Android developers who have published in the Play market, Section IV describes the design of our laboratory study, and Section V reports its results. In Section VI we present our analysis of Stack Overflow posts and in Section VII we present the ground truth from our static code analysis. Section VIII discusses some limitations of our work. Finally, in Section IX we discuss our results and conclude.

II. Related Work

In this section, we discuss related work in three key areas: Security and privacy flaws in otherwise benign mobile apps, efforts to understand how mobile developers make security- and privacy-relevant decisions and prior research exploring online Q&A resources such as StackOverflow.

Security Flaws in Mobile Apps. Many researchers attempted to measure the incidence of security flaws in otherwise benign mobile apps. Fahl et al. found that 8% of 13,500 popular, free Android apps contained misconfigured TLS code vulnerable to Man-In-The-Middle attacks [16]. Common problems included accepting all certificates without verifying their validity and not checking whether the name of the server currently being accessed matches the hostname specified on the certificate it provides. In follow-up work, the same research team extended their analysis to iOS and found similar results: Using a Man-In-The-Middle attack, they were able to extract sensitive data from 20% of the apps [18]. Another examination of TLS code, this time in non-browser software more generally, found similar flaws in many Android and iOS applications and libraries [20]. In more recent work, Onwuzurike and De Cristofaro found that the same problems remain prevalent several years later, even in apps with more than 10 million downloads [30]. Oltrogge et al. [29] investigated the applicability of certificate pinning in Android apps. They came to the conclusion that pinning was not as widely applicable as commonly believed. However, there was still a huge gap between developers who actually implement pinning and apps that could use pinning.

Egele et al. examined the use of cryptography – including block ciphers and message authentication codes – in Android applications and found more than 10,000 apps misusing cryptographic primitives in insecure ways [11]. Examples included using constant keys and salts, using non-random seeds and initialization vectors, and using insecure modes for block ciphers.

Many problems also exist with the use and misuse of app permissions, device identifiers, and inter-application communication. Enck et al. analyzed 1,100 free Android apps and reported widespread issues, including the use of fine-grained location information in potentially unexpected ways, using device IDs for fingerprinting and tracking (in concert with personal identifiable information (PII) and account registration), and transmitting device and location in plaintext [14]. Chin et al. characterized errors in inter-application communications (*intents*) that can lead to interception of private data, service hijacking, and control-flow attacks [9]. Felt et al. [33] analyzed how app developers use permissions and report that many request unnecessary permissions. The authors identify incomplete documentation for developers as one major root cause of this problem. Work by Poeplau et al. reported that almost 10% of analyzed apps load code via insecure channels (e.g., http or the SD card), which can allow attackers to inject malicious code into benign apps in order to steal data or create malware [31].

Enck et al. [13] presented TaintDroid – a tool that applies dynamic taint tracking to reveal how apps actually use permission-protected data. They uncovered a number of questionable privacy practices in apps and motivated enhancements to Android's original permission system and access control on

inter-component communication.

In this paper, we consider how the information sources developers use contribute to these kinds of errors and problems.

Understanding Developers. Many of the flaws discussed above arose from developer mistakes and misunderstandings. In interviews with developers who made mistakes in TLS code, Fahl et al. found that problems arose from several sources, including developers who disabled TLS functionality during testing and never re-enabled it, developers who did not understand the purpose of TLS or the possible threat scenarios, and problems with default configurations in frameworks and libraries [18]. Georgiev et al. also reported that confusion about the many parameters, options, and defaults of TLS libraries contributed to developer errors [20]. Both papers noted that developer forums such as Stack Overflow contained many suggestions for avoiding TLS-related error messages by disabling TLS features, without warning about the potential security consequences. Many developers use these resources to solve security- and privacy-related problems [3]. Similarly, Egele et al. discussed how poor default configurations and confusing APIs, along with insufficient documentation, may contribute to errors using cryptographic primitives [11].

In a non-mobile context, Leon et al. found that many popular websites used invalid or misleading P3P compact policies, which are tokens used to summarize a website's privacy policy for automated parsing [24]. Their manual analysis suggested that while many mistakes likely resulted from developer error, others resulted from attempts to avoid Internet Explorer's cookie filtering mechanism, and appeared to rely on suggestions from forums like Stack Overflow for avoiding this filtering. While these works on TLS and compact policies observed problems related to Stack Overflow and similar sites, our work uses a controlled experiment to compare the impact of different information sources.

Other flaws, particularly those related to privacy, are caused when developers do not sufficiently consider the implications of their decisions. In interviews with mobile developers from companies of various sizes, Balebako et al. found that privacy policies are not considered important and that privacy concerns are frequently outweighed by concerns about revenue, time to market, and the potential for any data that can be collected to someday be useful [2]. In a follow-up survey, the same authors found that many developers are not aware of the privacy or security implications of third-party advertising and analytics libraries they use [3]. These findings provide valuable insight into developers' perspectives; our work extends these perspectives with empirical observation of developer behavior.

Other researchers considered how to improve developers' decision making. Jain and Lindqvist propose a new location-request API designed to promote privacy-preserving choices by developers [21]. Fahl et al. suggested providing TLS as a service within a mobile OS that supports a separate development mode [18]. Similarly, Onwuzurike and De Cristofaro provided a code snippet for correctly using self-signed certificates during development but not production [30]. Our work extends Jain and Lindqvist's methodology to empirically evaluate developers' decisions.

Collectively, these findings suggest that helping well-meaning mobile developers to make better security- and privacy-relevant decisions could have a large impact on the overall mobile ecosystem. In this paper, we expand on these findings by using a controlled lab study to quantify how documentary resources impact security and privacy outcomes.

Exploring Online Q&A Resources. The software engineering and machine learning communities explored how developers interact with Stack Overflow and other Q&A sites. Much of this research focused on what types of questions are asked, which are most likely to be answered, and who does the asking and answering [5], [6], [27], [38]–[40].

Other research considered the quality of questions and answers available on Q&A sites – including general sites not specifically targeting programming [4], [22], [32]. These works are generally intended to support automated identification and pruning of low-quality content. In contrast to the studies described above, our work does not describe or measure broad trends in how Stack Overflow is used; nor do we consider how to automatically classify content. Instead, we directly consider how existing Stack Overflow content affects the outputs of developers who rely on it.

Others have analyzed Q&A sites specifically in the context of mobile development. Linares-Vásquez et al. investigated how changes to Android APIs trigger activities on Stack Overflow and found that the frequency of questions increases when Android APIs change, particular in the case of method updates [25]. In two related works, Wang et al. mined Stack Overflow posts to identify mobile APIs (Android and iOS) that frequently give developers trouble. They proposed that this data can be used both to improve documentation for these "hotspots" and to help API providers improve the design of their APIs to better support developer needs [41], [42]. In a similar vein, Nadi et al. analyze Stack Overflow posts to identify difficulties that developers commonly have with Java cryptography APIs [28]. While these works used Stack Overflow to identify trouble spots within APIs, we instead start from known trouble spots in security and privacy and measure how information sources, including Stack Overflow, directly affect the code developers write.

III. Survey of Android Developers

To understand the challenges app developers face during the implementation of security-critical app components, we conducted an online survey of Android developers covering their experience, their programming habits, and the resources they use. Results from this survey helped motivate the design of our lab experiment (Section IV). In this section, we briefly discuss the design of this survey as well as the results. The online study was approved by the University of Maryland Institutional Review Board.

We collected a random sample of 50,000 email addresses of Android application developers listed in Google Play (the official Android application market). We emailed these developers, introducing ourselves and asking them to take our

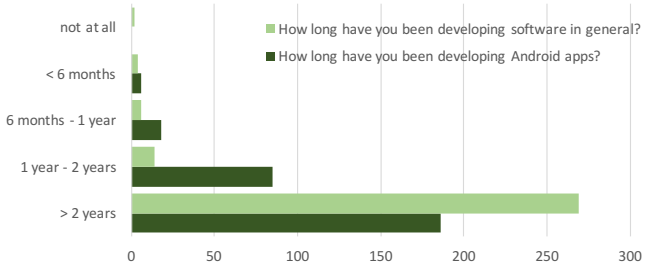

Fig. 1. How long participants in our online survey have been developing software, both in general and specifically for Android.

online survey. A total of 302 people completed the survey between April 2015 and October 2015. Seven participants were removed for providing answers that were nonsensical, profane, or not in English. Results are presented for the remaining 295.

Education and Experience. Most participants (91.2%, 269) had been developing software for more than two years; 63.1% (186) had been developing Android apps specifically for more than two years, as shown in Figure 1. About half of them (48.7%, 147) had developed between two and five apps; however, 73.5% (218) of all participants reported that they do not develop Android apps as their primary job.

Almost half of the participants had formally studied programming at the undergraduate (27.8%, 82) or graduate level (18.6%, 55). Most of the remaining developers reported being self-taught (41.2%, 121). Most participants had never taken any classes or training related specifically to Android programming (81.3%, 239) or to computer or information security (56.6%, 167).

As we discuss in Section V-A, these demographics have some similarity with our lab study participants; however, survey participants as a whole reported less formal education than lab participants.

Security and Permissions. We also asked participants about three security-related issues they might have encountered during app development: HTTPS/TLS, encryption, and Android permissions. These results provide some context for the security tasks used in our lab study.

About half of the developers (144) said that their Android app(s) use HTTPS to secure network connections; of those, 80.6% (116) had looked up information on HTTPS- or TLS-related topics at least once, but only 11.1% (16) did so more frequently than once per month. The most popular resources among these 116 were Stack Overflow (43.1%, 50) and a search engine such as Google (37.1%, 43); only 8.6% (10) mentioned the official Android documentation. Interestingly, a few (2.6%, 3) mentioned asking for help from certification-related companies such as certificate vendors or hosting companies. A large majority of respondents (78.4%, 91) said they did not handle HTTPS or certificate problems differently from

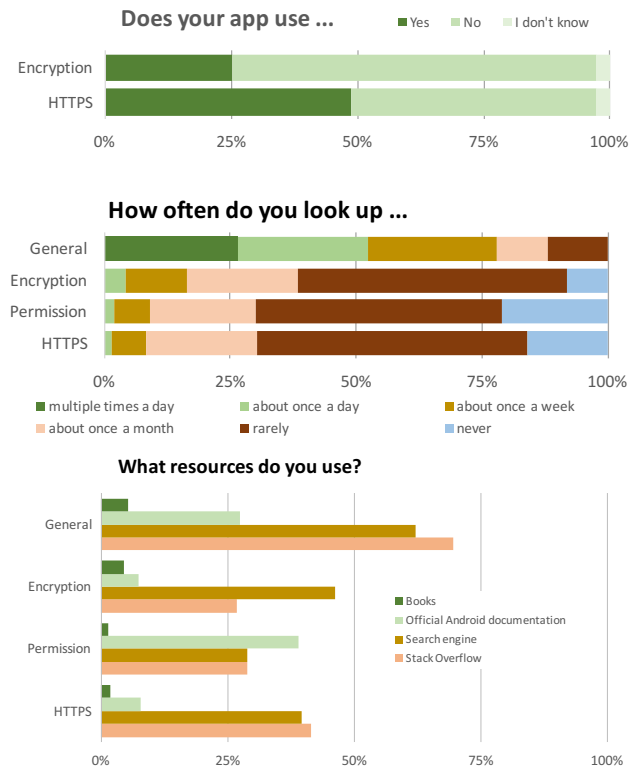

Fig. 2. Highlights of resource questions from our online developer survey. How many participants work on apps that include encryption or HTTPS (top), how often participants look up information when solving general programming problems or security-related Android problems (middle), how many participants mentioned using each of the most popular resources for solving general programming problems or security-related Android problems (bottom).

other problems.

Fewer participants (25.1%, 74) had used encryption to store files. Of these, almost all (90.5%, 67) had looked up encryption-related topics at least once, but again the vast majority did so once a month or less (82.1%, 55). The primary sources were once again search engines (mentioned by 31 participants, 46.3%) and Stack Overflow (28.4%, 19). Six of the 67 (9.0%) mentioned the official Android documentation, and two (3.0%) mentioned books. As with HTTPS, the majority (58, 86.6%) solved encryption problems similarly to other problems.

Responses to questions about Android permissions were somewhat different. As with HTTPS and encryption, most (74.9%, 221) reported they had looked up permissions information at least once, and a large majority of them did so once per month or less (84.2%, 186). However, participants who had looked up permission information favored official documentation (41.2%, 91) over search engines (29.0%, 64) or Stack Overflow (30.3%, 67) on that topic. One participant wrote that "[I] don't have to Google. [I] go directly to Android developer resource" for authoritative information.

Development Resources More Generally. We also asked (free response) about the resources participants use when

they encounter programming problems in general. The results are similar to those for security-specific problems. Large majorities mentioned Stack Overflow (69.5%, 205) and a search engine (62.0%, 183). Although this question did not specifically mention Android programming, 27.5% (81) also mentioned official Android documentation, including APIs and best practices guides.

In a separate question, we asked how frequently participants use any resources when programming for Android. More than half (52.2%, 154) reported looking up Android programming information at least once per day and another 25.4% said at least once per week. Among 35 participants (11.9%) who selected "rarely," 11 (31.4%) explicitly mentioned that while they rarely looked things up now, they had used resources or documentations for help many times a day when they were working on Android projects.

Figure 2 illustrates how participants used resources, both for security-related tasks and in general.

Discussion. Overall, these results indicate that many Android developers must deal with security or privacy issues periodically, but do not handle them consistently enough to become experts. This suggests that the quality of documentation is especially critical for these topics. Stack Overflow (and more generally, online search) is the default resource for certificate or encryption problems, as well as programming problems more generally. Permissions, however—perhaps because they are Android-specific and closely associated with the platform itself—are more frequently referenced from the official documentation. These findings validate both the need to understand the impact of the resources on security and privacy decisions generally, and our choice to compare Stack Overflow and the official documentation more specifically.

IV. Android Developer Study

To examine how the resources developers access affect their security and privacy decision-making, we conducted a between-subjects laboratory study. We provided a skeleton Android app and asked participants to complete four programming tasks based on the skeleton, encompassing the storage of data, the use of HTTPS, the use of ICC and the use of permissions. Each participant was assigned to one of four conditions governing what resources they were allowed to access. We examined the resulting code for functional correctness as well as for security- or privacy-relevant decisions; we also used a think-aloud protocol and an exit interview to further examine how participants used resources and how this affected their programming.

The lab study was also approved by the University of Maryland Institutional Review Board.

A. Recruitment

We recruited participants who had taken at least one course in Android development or developed professionally or as a hobby for at least one year. Initially, participants were also asked to complete a short programming task to demonstrate competence with Android development. After receiving feedback that the qualification task required too great a time commitment for prospective participants, we instead required participants to correctly answer at least three of five multiple choice questions testing basic Android development knowledge. The bar for qualification was intentionally set low, as we wanted to compare the impact of programming resources for developers with different expertise levels. In addition, the usefulness of our results partially depended on our participants needing to look things up during the programming process.

Participants were recruited in and around one major city in the U.S., as well as in two university towns in Germany. We recruited participants by emailing undergraduate and graduate students (in computer science in general and specifically those who had taken mobile development courses), as well as by placing ads on Craigslist, emailing local hacker and developer groups, and using developer-specific websites such as meetup.com. Prospective participants who qualified were invited to complete the study at a university campus or at another public place (library, coffee shop) of their choice. No mention of security or privacy was made during recruitment. Participants were compensated with $30 in the U.S. or an €18 gift card in Germany.

B. Conditions and study setup

Participants were assigned round-robin to one of four conditions, as follows:

Official Only (official). Participants were only allowed to access websites within the official Android documentation [2].

Stack Overflow Only (SO). Participants were only allowed to access questions and answers within Stack Overflow, a popular crowd-sourced resource for asking and answering questions about programming in a variety of contexts.

Book Only (book). Participants were only allowed to use two books: Pro Android 4 [23] and Android Security Internals [12]. Participants were provided access to the PDF versions of the books, enabling text searching as well as use of indices and tables of contents.

Free Choice (free). Participants were allowed to use any web resources of their choice, and were also offered access to the two books used in condition book.

Conditions official and SO were enforced using whitelist-chrome[3], a Chrome browser plugin for limiting web access.

Participants were provided with AndroidStudio, pre-loaded with our skeleton app, and a software Android phone emulator. The skeleton app, which was designed to reduce participants' workload and simplify the programming tasks, was introduced as a location-tracking tool that would help users keep track of how much time they spent in various locations (at home, at work, etc.) each day.

After a brief introduction to the study and the skeleton app, participants were given four programming tasks in random order (detailed below), with approximately 20-30 minutes to

[2] cf. http://developer.android.com

[3] https://github.com/unindented/whitelist-chrome

complete each. (The first task was allowed to run longer as participants became acquainted with the skeleton app.) While the short time limit impeded some participants' performance, it also simulated the pressure of writing code on tight deadlines that many app developers face.

Security and privacy were not mentioned during the introduction to the study and skeleton app or in the directions for each task (the HTTPS task and password task do inherently imply some reference to security). We deliberately minimized security priming to account for the fact that security and privacy are generally secondary tasks compared to basic app functionality [2], [10], [18], [19]. Instead, we focus on whether developers – who in real-world scenarios may or may not be explicitly considering security – find and implement secure approaches. This is in line with prior studies examining security and privacy decisionmaking by developers, such as one by Jain and Lindqvist [21].

C. The Tasks

Each participant was assigned the same four tasks, but in a random order. We took care to implement baseline functionality so that the tasks could be done in any order and so that remaining tasks would still work, even if a previous task had not been successfully completed.

Drawing on related work (as discussed in Section II), we selected four general areas that typically result in security or privacy errors on Android: (1) Mistakes in TLS and cryptographic API handling; (2) storing sensitive user data insecurely, such that it can be accessed by other (unauthorized) apps; (3) using inter-component communications (ICC) in a way that violates least privilege principles; and (4) requesting unneeded permissions. We designed four tasks, detailed below, to exercise these areas respectively.

Secure Networking Task. This task addressed correct usage of HTTPS and TLS in the presence of X.509 certificate errors. The skeleton app connected to a website via HTTP; participants were asked to convert the connection to HTTPS. This required making a minor adjustment to the connection object. More interestingly, we created a certificate for secure.location-tracker.org (a server we configured specifically for this study), but the participant was requested to connect to location-tracker.org, and a matching DNS entry for secure.location-tracker.org did not exist. As a result, participants received a HostnameVerifier exception indicating the certificate name and domain were mismatched. Secure solutions would include creating a custom HostnameVerifier to handle this case or pinning the certificate (although we expected pinning to be too time-consuming for most participants to implement in the study) [4]. We also accepted a participant arguing that the location tracker app should obtain a correct X.509 certificate rather than working around the problem as a secure solution. Insecure solutions that allow a connection to be established include using a HostnameVerifier that accepts all hostnames, or simply accepting all certificates without validation.

[4]Implementing it correctly requires inspecting the server's certificate and using a third-party tool such as the OpenSSL command-line client to generate the pinning information

ICC Task. This task addressed secure inter-component communication. Participants were asked to modify a service within the skeleton app, in order to make the service callable by other apps. However, participants were asked to limit this access to apps created by the same developer. To accomplish this, participants needed to modify the Android Manifest. An insecure solution would expose the service to be called by any app; this could happen by setting the flag android:exported to true or by declaring intent filters, which set the exported flag to true by default. A secure solution for this task is to define an own permission with protection level 'signature' or 'signatureOrSystem' and assign it as required for the service. A second possible secure solution is to use a sharedUserId among all apps from the same developer, which allows the apps to share resources.

Secure Storage Task. This task focused on secure storage of the user's login ID and password for the remote server. The skeleton app contained empty store and load functions for the participant to fill in; the directions asked the participant to store the credentials persistently and locally on the device. A secure solution would be to limit access only to this app, for example using Android's shared preferences API in private mode. An insecure solution would make the data accessible to third parties, for example by storing it world-readable on the SD card.

Least Permissions Task. In this task, participants were asked to add functionality to dial a hard-coded customer support telephone number. The skeleton app contained a non-functional call button, to which the dialing functionality was to be applied. To solve this problem, the participant needed to use an intent to open the phone's dialing app. One option is to use the ACTION_DIAL intent, which requires no permissions; it opens the phone's dialer with a preset number but requires the user to actively initiate the call. Another option is to use the ACTION_CALL intent, which initiates the call automatically but requires the CALL_PHONE permission. We consider the second solution less appropriate because it requires unnecessary permissions, violating the principle of least privilege.

D. Exit Interview

After completing each task (or running out of time), participants were given a short exit interview about their experience. Using a five-point Likert scale, we asked whether each task was fun, difficult, and whether the participant was confident they got the right answer. We also asked whether the documentation and resources participants had access to were easy to use, helpful, and correct. We asked free-response questions about whether the participant had used that documentation source before and how they felt the documentation restriction (where applicable) and time crunch affected their performance. We also asked whether and how participants had considered security or privacy during each task. Finally, we asked a series

of demographic and programming-experience questions that matched those in our initial developer survey (see Section III).

E. Data Collection and Analysis

In addition to each participant's code, think-aloud responses, and exit interview responses, we collected browser activity during the session (for participants in all but the book condition) using the History Export[5] extension for Chrome, which stores all visited URLs in a JSON file.

Scoring the Programming Tasks. For each programming task, we assigned the participant a *functionality* score of 1 (if the participant's code compiled and completed the assigned task) or 0 (if not). To provide a security score for each task, we considered only those participants who had functional solutions to that task. We manually coded each participant's code to one of several possible strategies for solving the task, each of which was then labeled secure or insecure. Based on these categories, each participant who completed a task was assigned a security score of 0 (insecure approach) or 1 (secure approach) for that task. Manual coding was done by two independent coders, who then met to review the assigned codes and resolve any mismatches. All conflicts were resolved by discussions that resulted in agreement. Example secure and insecure approaches for each task are detailed in Table I.

Prior to the conducting the lab study, we verified that functional and secure solutions for each task, such as those described in Table I, were available in each of the official Android documentation, Stack Overflow, and the selected books. This ensured that it was possible (if not necessarily easy) for participants in all conditions to locate a correct and secure answer.

Statistical Methods. For ordinal and numeric data, we used the non-parametric Kruskal-Wallis test to compare multiple samples and Wilcoxon Signed-Rank test to compare two samples. For categorical data, we used Fisher's Exact test. In cases of multiple testing, we report tests as significant if the p-values are significant after applying the Bon Ferroni-Holm correction. To examine correlation between two sets of binary outcomes, we the use Cohen's κ measure of inter-rater reliability.

To examine functional correctness and security across tasks and conditions, while accounting for multiple tasks per participant, we used a cumulative-link (logit) mixed model (CLMM) [45]. As is standard, we include random effects to group each participant's tasks together. For the CLMM, we tested models with and without the participant's status as a professional developer as an explanatory factor, as well as with and without interactions among task, condition, and developer status. In each case, we selected the model with the lowest Akaike information criterion (AIC) [7].

V. Lab Study Results

In this section, we discuss our lab study results in terms of functional correctness, security, and participants' use of their assigned resources. We find that while Stack Overflow is easier to use and results in more functional correctness, it also results in less security than the less accessible official API documentation.

[5]https://chrome.google.com/webstore/detail/history-export/lpmoaclacdaofhlijejogfldmgkdlglj

Assigned condition			
Official: 13	SO: 13	Book: 14	Free: 14
Location of Study			
United States: 12 (22.2%)			Germany: 42 (77.8%)
Gender			
Male: 46 (85.2%)			Female: 8 (14.8%)
Country of Origin			
United States: 6 (11.1%)			Germany: 28 (51.9%)
India: 5 (9.3%)			Others: 15 (27.8%)
Professional Android Experience			
Yes: 14			No: 40
Ages			
mean = 26.0	median = 25		sd = 4.7

TABLE II
Participant Demographics.

A. Participants

A total of 56 people participated in our lab study (13 in the U.S. and 43 in Germany). Two participants (one from the U.S. and one from Germany) were removed, one due to an error assigning the condition and one because of their refusal to work on the tasks. We report results for the remaining 54.

Our participants were aged between 18 and 40 (mean = 26, sd = 4.70), 85.2% were male (46 participants), and most of them (88.9%, 48) were students. Several were part-time students and part-time professional developers. More than half of participants said they grew up in Germany (51.9%, 28). The next most popular countries of origin were the U.S. (11.1%, 6) and India (9.3%, 5). Table II shows demographic information for the participants recruited in each country. Using Fisher's exact test, we did not find differences in gender ($p = 0.400$), occupation ($p = 1.00$) or country of origin ($p = 0.81$) between the randomly assigned conditions. Using the Kruskal-Wallis test, we could not find a difference in ages across the randomly assigned conditions ($X^2 = 2.22$, $p = 0.528$). Both in the U.S. and in Germany, participants were distributed evenly across the four conditions.

Every lab study participant but one (98.1%) had been programming in general for more than two years; 51.9% (28) had been specifically developing Android apps for more than two years. About half of the participants (53.7%, 29) had developed between two and five Android apps, and 18.5% (10) had developed 10 or more apps. Most participants (85.2%, 46) were not developing Android apps as their primary job, but eight participants were employed as Android app programmers. Using the Kruskal-Wallis test, we did not find a difference in years of Android experience or in number of apps developed across the randomly assigned conditions ($X^2 = 5.06, 4.46$ and $p = 0.409, 0.485$ respectively). As shown in Figure 3, our lab-study participants had roughly similar experience to the developers in our online survey.

Task	API	Details	Security Rating	Explanation
Secure Networking	`javax.net.ssl.HostnameVerifier.verify(host, session)`	`return true`	○	A custom hostname verifier with a correct domain check is rated as a secure solution. Hostname verifiers which accept all hostnames are rated insecure.
		e.g. `host.equals("secure.foo.com")`	●	
	`org.apache.http.conn.ssl.X509HostnameVerifier.verify(host, session)`	`return true`	○	
		e.g. `host.equals("secure.foo.com")`	●	
ICC	`<service>...</service>`	`android:exported=true`	○	A service that has the exported flag set to true, uses intent filters, or uses a normal or dangerous permission is rated as insecure. Services protected with a signature or signatureOrSystem permission are rated as secure.
		`<intent-filter>...</intent-filter>`	○	
		`android:permission`	○	
		`android:protectionLevel=signature`	●	
		`android:protectionLevel=signatureOrSystem`	●	
Secure Storage	`Environment.getExternalStoragePublicDirectory(type)`	–	○	We distinguish three different storage backends: SQLite databases, the file system, and Android's shared preferences. All three can have secure and insecure implementations. Secure implementations store information in an area local to an app; this is the default implementation for SQLite databases and shared preferences. However, both backends can be used to store data such that other apps can access it. The file-system API can be used to either store data locally or externally on a device's SD card. Implementations that store information in an externally accessible way are rated insecure. Implementations that store information locally are rated secure.
	`Context.getExternalFilesDir(type)`	–	○	
	`Context.getFilesDir()`	–	●	
	`Context.getCacheDir()`	–	●	
	`Context.openFileOutput(name, mode)`	`Context.MODE_PRIVATE`	●	
		`Context.MODE_WORLD_READABLE`	○	
		`Context.MODE_WORLD_WRITABLE`	○	
	`Context.getDir(name, mode)`	`Context.MODE_PRIVATE`	●	
		`Context.MODE_WORLD_READABLE`	○	
		`Context.MODE_WORLD_WRITABLE`	○	
	`PreferenceManager.getDefaultSharedPreferences()`	–	●	
	`PreferenceManager.getSharedPreferences(context)`	–	●	
	`Context.getSharedPreferences(name, mode)`	`Context.MODE_PRIVATE`	●	
		`Context.MODE_WORLD_READABLE`	○	
		`Context.MODE_WORLD_WRITABLE`	○	
	`Activity.getPreferences(mode)`	`Context.MODE_PRIVATE`	●	
		`Context.MODE_WORLD_READABLE`	○	
		`Context.MODE_WORLD_WRITABLE`	○	
	`Context.openOrCreateDatabase(name, mode,...)`	`Context.MODE_PRIVATE`	●	
		`Context.MODE_WORLD_READABLE`	○	
		`Context.MODE_WORLD_WRITABLE`	○	
Least Permissions	`new Intent(action, uri)`	`Intent.ACTION_DIAL`	●	Using action_dial is rated secure. However, using action_call and requesting the call_phone permission is rated insecure.
		`Intent.ACTION_CALL`	○	
		`android:name='android.permission.CALL_PHONE'`	○	

● = we rated this solution as secure, ○ = we rated this solution as insecure

TABLE I
TASK RELATED API CALLS AND THEIR PARAMETERS. WITH SECURITY RATING PARAMETERS HELP CLASSIFY WHETHER A SOLUTION IS SECURE.

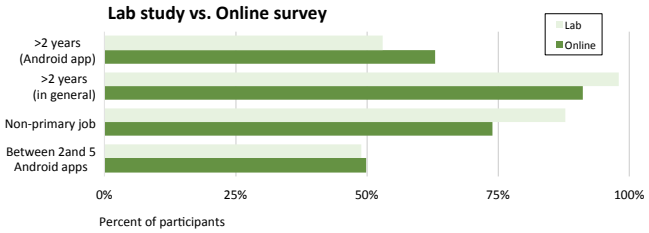

Fig. 3. Comparison of programming experience for participants in our online survey and lab study.

We also asked how participants learned to program (multiple answers allowed). Almost all (83.3%, 45/54) said they were at least partially self-taught, and 79.6% (43) had formally studied programming at the undergraduate or graduate level. More than half (63.0%, 34) had taken at least one security class, and slightly fewer than half (46.3%, 25) had taken a class in Android programming. Overall, our lab study participants had notably more education than the developers in our online survey.

B. Functional Correctness Results

Our results demonstrate that the assigned resource condition had a notable impact on participants' ability to complete the tasks functionally correctly; SO and book participants performed best, and official participants performed worst. SO participants solved 67.3% (35/52) of tasks correctly, compared to 66.1% (37/56) for book, 51.8% (29/56) for free, and 40.4% (21) for official. Figure 4 (top) provides more detail on the breakdown of correctness across tasks and conditions. The CLMM model (see Table III) indicates that when controlling for task type, professional status, and multiple tasks per participant, participants in official were significantly less likely than baseline SO participants to functionally complete tasks.

Factor	Coef.	Exp(coef)	SE	p-value
free	-1.054	0.349	0.613	0.085
official	-1.535	0.215	0.634	0.015*
book	-0.142	0.868	0.602	0.814
ICC	0.795	2.215	0.455	0.081
secure storage	1.280	3.597	0.468	0.006*
least permissions	3.299	27.092	0.632	< 0.001*
professional	1.004	2.728	0.501	0.045*

TABLE III
CLMM REGRESSION TABLE FOR FUNCTIONAL CORRECTNESS. THE NON-INTERACTION MODEL INCLUDING PROFESSIONAL STATUS WAS SELECTED. NON-SIGNIFICANT VALUES ARE GREYED OUT; SIGNIFICANT VALUES ARE INDICATED WITH AN ASTERISK. THE BASELINE FOR CONDITION IS SO, AND THE BASELINE FOR TASK IS SECURE NETWORKING.

Participants' perceptions of the tasks only partially dovetailed with these results. We asked participants, on a 5-point Likert scale, whether they were confident they had gotten the right answer for each task.[6] Participants in condition free were most confident: They agreed or strongly agreed they were confident for 55.4% of tasks. Participants in each of the other three conditions were confident for slightly fewer than half of tasks: 47.3% in book and 46.2% in both SO and official. Figure 5 illustrates these results.

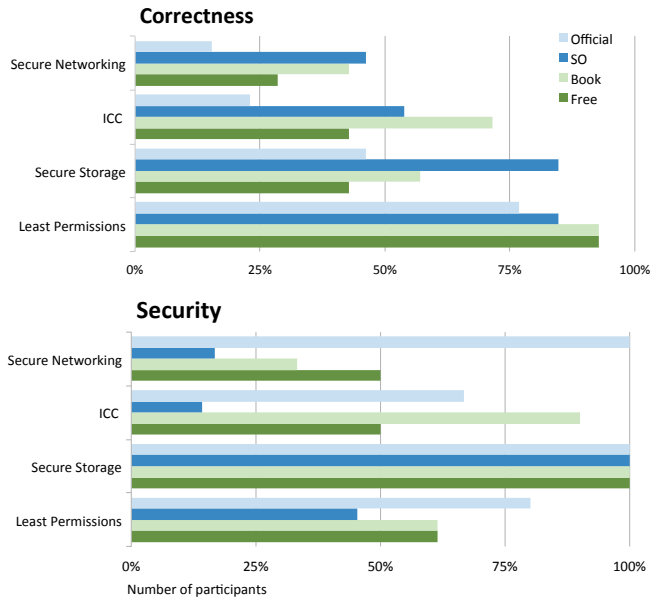

Fig. 4. Top: Percentage of participants who produced functionally correct solutions, by task and condition. Bottom: Percentage of participants whose functionally correct solutions were scored as secure, by task and condition.

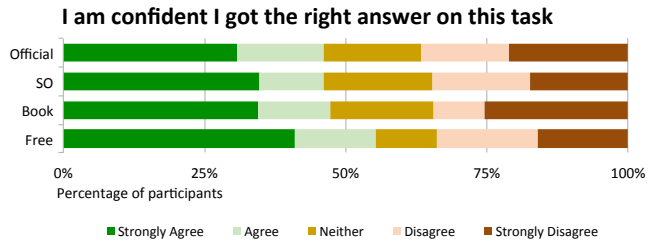

Fig. 5. Participants' confidence in their tasks' correctness, by condition, on a 1-5 Likert scale (1 = Strongly disagree, 5 = Strongly agree).

Using Cohen's κ, we examined whether participants' self-reported confidence in their tasks' correctness (binned as strongly agree/agree and strongly disagree/disagree/neutral) matched with our functional correctness result. We found $\kappa = 0.55$, indicating that participants were assessing their functional correctness only somewhat effectively.

Correctness per Task. Observed correctness varied strongly among the four tasks, as shown in Figure 4 (top). In the least permissions task, 87.0% (47) of participants produced

[6]One book participant's confidence answer for the least permissions task was inadvertently not recorded; we exclude that participant from confidence analyses only.

a functional solution; in the secure networking task only 33.3% (18) did. These results were mirrored in self-reported confidence: 81.1% of participants were confident about the least permissions task, compared to 53.7% for secure storage, 40.7% for ICC, and only 20.1% for secure networking. The CLMM analysis (Table III) indicates that both the secure storage and least permissions tasks were significantly more likely to be functionally correct than the baseline secure networking task.

C. Security Results

Our results suggest that choice of resources has the opposite effect on security than it did on functionality: SO participants performed worst on this metric. As described in Section IV-E, we scored tasks that had been solved correctly for security, privacy, and adherence to least-privilege principles. In the SO condition, only 51.4% (18/35) of functional solutions were graded as secure, compared to 65.5% (19/29) for free, 73.0% (27/37) for book, and 85.7% (18/21) for official. Figure 4 (bottom) illustrates these results. Using a CLMM that includes only tasks that were graded as functionally correct (Table IV), we find that both official and book produce significantly more secure results than SO. The difference between SO and free, in which many participants elected to use Stack Overflow for most of their tasks (see Section V-D), was not significant.

Factor	Coef.	Exp(coef)	SE	p-value
free	1.112	3.040	0.623	0.074
official	2.218	9.184	0.796	0.005*
book	1.539	4.660	0.604	0.011*
ICC	0.763	2.144	0.666	0.252
least permissions	0.856	2.353	0.609	0.160

TABLE IV
CLMM REGRESSION TABLE FOR SECURITY. ONLY TASKS GRADED AS FUNCTIONALLY CORRECT ARE INCLUDED IN THE MODEL. THE NON-INTERACTION MODEL WITHOUT PROFESSIONAL STATUS WAS SELECTED. NON-SIGNIFICANT VALUES ARE GREYED OUT; SIGNIFICANT VALUES ARE INDICATED WITH AN ASTERISK. THE BASELINE FOR CONDITION IS SO, AND THE BASELINE FOR TASK IS SECURE NETWORKING.

Security per Task. As with correctness, security results differed noticeably among tasks. For example, every participant who produced a functional solution to the storage task (31) produced a secure solution. On the other hand, only 38.9% (7/18) of participants who produced a functional solution to the networking task were scored as secure. This discrepancy is illustrated in Figure 4 (bottom). Our CLMM results (Table IV) indicate that neither the ICC nor least permissions task was significantly different from the networking task. Because all functional solutions to the storage task were graded as secure regardless of condition, the regression coefficient for that task approaches infinity, and the results of the model estimates for that task are not interpretable. We omit it from the table.

Considering Security while Programming. We were also interested in the extent to which participants thought about security while solving each task. We measured this in two ways. First, we considered the participants' think-aloud comments for each task, classifying them as having either explicitly mentioned security; mentioned security but later decided to ignore it for the task at hand; or not mentioned security at all. These classifications were independently coded by two coders who then reached agreement, as discussed in Section IV-E. We refer to this as *observed* security thinking. Second, we asked participants during the exit interview to self-report for each task whether or not they had considered security, as a yes/no question. We refer to this metric as *self-reported* security thinking. For both metrics, we considered all tasks, not just those that participants completed correctly.

In the observed metric, most participants did not mention security at all (79.2% of all tasks, 171). In the storage task, 16 participants (29.6%) mentioned security and all stuck with it; in the networking task 20 mentioned security (37.0%) but nine later abandoned it. In contrast, only five and four participants ever mentioned security or privacy in the least permissions and ICC tasks, respectively. Common reasons for abandoning security included that finding a secure solution proved too difficult, that the task was for a study rather than real, and that the participant was running short of time.

In the self-reported metric, more participants reported considering security: 60.2% of all tasks (130). Using this metric, security was most frequently considered for secure networking (79.6%), followed by ICC (70.4%) and secure storage (68.5%). Only 22.2% of participants reported considering security for the least permissions task. The higher rate of security thinking using this metric is most likely attributable to the participants being prompted.

To compare conditions, we assign each participant a separate score for each metric, corresponding to the number of tasks in which the participant considered security. In both metrics the average scores were highest in book (0.93, 2.86) and lowest in SO (0.69, 1.92), but neither difference was significant (Kruskal-Wallis, observed: $X^2 = 0.507$, $p = 0.917$, self-report: $X^2 = 4.728$, $p = 0.192$).

Comparing Professionals and Non-Professionals. Although the relatively small sample of professionals we were able to recruit makes comprehensive comparisons difficult, we examined differences in correctness and security between these two groups. For purpose of this analysis, we categorize 14 participants as professionals, including those who are currently or recently had been professional developers. The non-professional participants are primarily university students. The professionals were randomly distributed across conditions: five in free, three in SO, two in official and four in book.

Overall, professionals were slightly more likely to produce a functional solution, with a median three functionally correct tasks (mean = 2.79, sd = 0.70) compared to two functionally correct tasks (mean = 2.08, sd = 1.23) for non-professionals. We observed essentially no difference in security results: professionals' solutions were median 66.7% secure (mean = 69.0%, sd = 0.20), compared to 66.7% for non-professionals (mean = 66.2%, sd = 0.36). These observations fit with the CLMM results: professional status predicts a small but

significant increase in functional correctness, but professional status is excluded from the best-fitting security model.

D. Use of Resources

During the tasks, we collected the search terms used and pages visited by all participants in non-book conditions. In addition, during the exit interview, we asked participants several questions about the resources they were assigned to use. In this section, we analyze participants' search and lookup behavior as well as their self-reported opinions.

Lookup Behavior Across Conditions.

We define "search queries" as submitting a search string to a search engine or to the search boxes provided by Stack Overflow and the official Android documentation. Participants in the SO condition made on average 22.8 queries across the four tasks, compared to 14.5 for the official condition. Participants in free were closer to SO than official, with an average of 21.1 queries. We also observed that participants in the official, free and SO conditions visited on average 35.4, 36.1, and 53.2 unique web pages across the four tasks. We offer two hypotheses for these results, based on on our qualitative observations: First, official participants were more likely to scroll through a table of contents or index and click topics that seemed relevant (as opposed to doing a keyword search) than those in other conditions, presumably because the official documentation is more structured. Second and perhaps more importantly, SO participants seemed to be more likely to visit pages that turned out to be unhelpful and restart their searches.

Participants in the free condition were given their choice of Internet resources to help them solve the programming tasks. Every free participant started every attempt to get help with a Google search. Undoubtedly this was partially influenced by Chrome using Google as the default start page as well as automatically using Google search for terms entered in the URL bar, but the complete unanimity (along with results from the online survey) suggests that many or most attempts would have started there anyway. From within their Google results, every participant selected at least one page within the official Android API documentation, and all but one visited Stack Overflow as well. A few visited blogs, and one visited an online book. These results are consistent with the online survey results reported in Section III. In terms of frequency, official documentation was most popular, representing between 50 and 85% of non-google-search pages for all participants except one outlier who visited it 98% of the time. Most participants visited Stack Overflow for between 10 and 40% of their pages, with outliers at 0 and 2.4% as well as 50%. While participants in the group visited more official documentation pages than pages at Stack Overflow, their functionality and security results more closely resemble the group than the official group. This may be partially explained by a behavior pattern that we observed several times in the free condition: participants spent some time reading through the official documentation, but as the time limit approached used content (often a copied and pasted code snippet) from Stack Overflow.

Search Query Selection. We also examined the search query text chosen by participants. Queries were normalized for capitalization and spacing, and any queries within one string edit of each other were consolidated (to account for plurals and typos). Because few participants exactly duplicated one another's queries, in order to discern trends, one researcher manually coded similar terms into categories. For example, "restrict access developers," "restrict app access for same developer," and "restrict apps same developer" were categorized together. For the secure networking task, the most common queries involved hostname exceptions and HTTPS, together with just a few searches for certificates, certificate errors, and hostname verifiers. For the ICC task, the most popular searches included manifest, permissions, services, external access, and restricting access. A few more knowledgeable participants searched for intent filters, user IDs, and signatures. For secure storage, the most popular choices included storage, persistent storage, and shared preferences; for least permissions participants most frequently searched for call and phone call, with a few searching for dial. Only four participants searched for "secure" or "security," including two in free and one each in SO and official.

Participants' Opinions about Information Sources. We asked our non-free participants whether they had previously used their assigned resource. All 14 SO participants had previously used Stack Overflow, and most (10/13) official participants had used the official documentation. However, only six of 14 book participants had used books. We also asked participants to rate, on a five-point Likert scale, the extent to which the resources they used were easy to use, helpful, and correct. Results are shown in Figure 6. As might be expected, participants found free choice easiest to use and books least easy; in contrast, they were most likely to consider books and the official documentation to be correct.

We also asked about the effect of participants' assigned resource on their performance. In every non-free condition, the large majority (official: 92.3% (12/13); book: 92.9% (13/14); SO: 78.6% (11/14)) said they would have performed better on the tasks if they had been allowed to use different resources. In particular, official and book participants said they would have liked to access Stack Overflow or search engines such as Google, so that they could search for their specific problems rather than reading background information. One book user mentioned the "danger that books could be outdated." On the other hand, many SO participants said they would have liked to access the official documentation to read up on background information for their problems.

Time constraints were also a concern for our participants. Most (61.1%, 33) said they would probably have performed better had they been given more time, while nine (16.7%) mentioned (unprompted) that more time would have allowed them to make their solutions more secure. One participant in official, for example, said that "Twenty minutes is very limited to consider security." The remaining 38.9% said more time would not have helped, either because they solved the tasks to their satisfaction, or because they believed the resource they

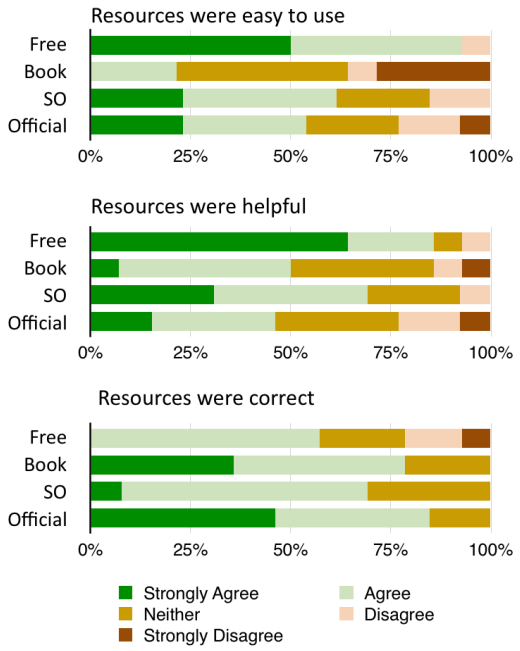

Fig. 6. Participants' agreement (on a five-point Likert scale) with the statements that the resources they used were easy to use, helpful, and correct, by condition.

were using did not allow them to find a (better) solution.

VI. Quality of Stack Overflow Responses

To better contextualize the performance of the participants – in both, the SO and free condition –, we examined in detail all Stack Overflow pages (threads) visited by our participants during the programming tasks. In particular, we were curious about whether these pages contained secure and/or insecure examples and code snippets, and whether the security implications were explained. As might be expected, we found many discouraging instances of insecure examples and few discussions of security implications.

A. Classification Methodology

We rated each thread on five different attributes, described below. All threads were independently coded by two researchers, who then reached consensus on any conflicts.

Task Relevance. We first checked whether the topic of the thread was actually relevant to solving the study task. If it would not help the participant in solving the task, it was flagged as off-topic and not looked at further.

Usefulness. We rated each on-topic thread as useful or not useful, based on how related answers were to the question. Threads with no answers, or no answers that responded to the original question, were rated as not useful. Threads with answers that discussed the question and gave helpful comments, links to other resources, or sample code were rated as useful.

Code Snippets. We examined all answers in each thread for ready-to-use code snippets. We rated a code snippet as ready-to-use if it was syntactically correct and a developer could copy and paste it into an app. Each thread in which at least one answer qualified was marked as containing a code snippet. Each code snippet was individually rated as secure or insecure relative to the programming tasks described in Section IV-C.

External Links. Within each thread, we looked for answers containing external links. We classified threads as containing links to GitHub, to other code repositories, to other Stack Overflow threads, or to anywhere else. Additionally, we classified the linked content as either secure or insecure.

Security Implications. We investigated whether any answer in the thread discussed security implications of possible solutions. For example, if two solutions existed and one included an extra permission request, we checked whether any of the answers discussed a violation of the least-privilege principle. If an answer contained a NullHostnameVerifier, we would check if at least one of the answers would advise that verification should not be disabled.

B. Classification Results

Overall, our participants accessed 139 threads on Stack Overflow. We categorized 41 threads as being on-topic. Table V summarizes the classification results for these 41 threads. Of these, 20 threads contained code snippets. Half of the threads containing code snippets contained only insecure snippets, such as instructions to use NullHostnameVerifiers and NullTrustManagers, which will accept all certificates regardless of validity. Among these 10 threads containing only insecure code snippets, only three described the security implications of using them. This creates the possibility for developers to simply copy and paste a "functional" solution that voids existing security measures, without realizing the consequences of their actions. More encouragingly, seven of the 10 remaining threads with code snippets contained only secure code snippets.

We next investigated how threads with different properties compared in terms of popularity (measured by total upvotes for the thread). Unsurprisingly, we found that threads with code snippets were more popular than those without ($W = 319.5$, $p = 0.00217$, $\alpha = 0.025$, Wilcoxon-Signed-Rank Test (B-H)). Discouragingly, we found no statistical difference between threads with secure and insecure code snippets ($W = 73$, $p = 0.188$). On the other hand, threads that discuss security implications are slightly more popular than those that don't ($W = 239.5$, $p = 0.0308$, $\alpha = 0.05$ (B-H)).

Although these results cover only a very small sample of Stack Overflow threads, they provide some insight into why our SO participants had lower security scores than those in the official condition.

VII. Programming Task Validity

To provide evidence for the validity of our lab study tasks and results, we examined how the APIs used in our programming tasks (cf. Table I) are used in real-world apps. In particular, we were interested in how frequently these APIs are used in real Google Play apps, as well as whether secure or

Answers in the thread include ...	Count	
Useful answers	35	(85.4%)
Useless answers	6	(14.6%)
Discussion of security implications	12	(29.3%)
Working code examples	20	(48.8%)
Only secure code examples	7	(17.0%)
Only insecure code examples	10	(24.4%)
...but also discussion of security implications	3	(30.0%)
Secure links	23	(56.1%)
Insecure links	6	(14.6%)
Links to GitHub	4	(9.8%)
Links to other code repositories	1	(2.4%)
Links to other Stack Overflow threads	4	(9.8%)
Only secure code examples and secure links	3	(7.3%)

TABLE V
PROPERTIES OF THE 41 ON-TOPIC STACK OVERFLOW THREADS ACCESSED DURING THE LAB STUDY.

Stack Overflow Threads			
with code snippets		**without code snippets**	
mean	97.7	mean	3.9
median	12	median	2.5
sd	163.9	sd	4.4
W = 319.5, p = 0.00217, α = 0.025 (B-H)			
with secure code snippets		**with insecure code snippets**	
mean	204.3	mean	70.2
median	145	median	14
sd	209.3	sd	122.4
W = 73, p = 0.188			
with security implications		**without security implications**	
mean	135.2	mean	17.4
median	16	median	3
sd	207	sd	37
W = 239.5, p = 0.0308, α = 0.05 (B-H)			

TABLE VI
POPULARITY RATINGS FOR THREADS CONTAINING CODE SNIPPETS.

insecure solutions are more prevalent. Results of our analysis show that the APIs we examined are widely used; in line with our lab study results, the secure networking and ICC APIs were frequently used in ways that suggest security concerns.

A. Analysis

To analyze real-world apps, we applied standard static code-analysis techniques: We decompiled Android APK files, constructed control flow graphs (CFGs), and applied backtracking to gather insights about how often real-world developers use APIs relevant to our programming tasks. Limitations of this approach are discussed in Section VIII. Overall, we analyzed a random sample of 200,000 free Android apps from Google Play.

Secure Networking Task. For this task, we analyzed whether an app implements the `HostnameVerifier` interface (cf. Table I). Hostname verification requires a developer to implement the `verify(String hostname, SSLSession session)` method. We checked if an implementation actually performs hostname verification by processing the `hostname` parameter or if it simply accepts every hostname (i. e. `return true;`).

ICC Task. For this task, we analyzed an app's Manifest file (cf. Table I). We extracted `<service>` entries from the XML DOM, then checked for `<intent-filter>` child nodes to determine whether an intent filter was set. We also checked whether the `android:exported` flag, which indicates whether a service is made publicly available, was present and if it was set to `true`. Lastly, we extracted `android:permission` attributes to see if services were protected by permissions. We also extracted the `android:protectionLevel` attributes to learn whether signature or system permissions are required to use this service.

Secure Storage Task. To determine whether an app stores data persistently, we looked up relevant API calls in the call graph. We distinguished between three different targets: SQLite databases, the file system, and shared preferences (cf. Table I).

To check for SQLite database usage, we looked up the `openOrCreateDatabase` API call in the CFG. Developers can use this API call in a way that keeps data local to an app by explicitly setting the `MODE_PRIVATE` flag or using the default. Setting the `MODE_WORLD_WRITEABLE` or `MODE_WORLD_READABLE` flag stores the database outside an app's local storage and makes it available to other apps. We used backtracking to check which flags were set.

To analyze file-system access, we looked up API calls that return output- or inputstreams to a file handle. This includes the `openFileOutput` method and the mode flags. Additionally, we checked for use of methods that find the path of the external storage as well as the `WRITE_EXTERNAL_STORAGE` permission.

To check for shared preferences usage, we looked up the `getSharedPreferences`, `getPreferences` and `getDefaultSharedPreferences` API calls in the CFG. The `MODE_PRIVATE`, `MODE_WORLD_WRITEABLE` and `MODE_WORLD_READABLE` flags are used to distinguish between secure and insecure solutions.

Least Permissions Task. To examine use of dialing compared to calling, we analyzed the Manifest file for the occurrence of the `CALL_PHONE` permission request and searched for relevant API calls in the CFG. To initiate a phone call, a new Intent object must be created using a string parameter to specify the intended action. We used backtracking to obtain the respective action value and searched for `ACTION_DIAL` and `ACTION_CALL` values.

Apps that used an `ACTION_DIAL` intent were rated as adhering to least privilege since they use the system's dialer and do not require an additional permission. Apps that use an `ACTION_CALL` intent in combination with requesting the `CALL_PHONE` permission were rated as not adhering to least privilege.

	secure	apps
Secure Networking Task		
broken hostname verifier	○	19,520
alternative hostname verification	●	214
ICC Task		
service	-	42,193
intent filter	○	8,133
exported=true	○	3,796
permission	◐	3,827
permission, signature	●	86
permission, signature or system	●	15
Secure Storage Task		
filesystem, private	●	120,834
filesystem, public	○	34,183
database, private	●	4,471
database, public	○	154
shared preferences, private	●	130,408
shared preferences, public	○	17,848
Least Permissions Task		
dial, permission	○	3,907
dial, no permission	●	48,832
call, permission	○	5,336
call, no permission	◐	6,157

● = secure; ○ = insecure

TABLE VII
RESULTS OF STATICALLY ANALYSING A RANDOM SAMPLE OF 200,000 ANDROID APPS.

B. Results

Table VII summarizes the results of our real-world app analysis, which are further detailed below.

Secure Networking Task. We identified 19,734 apps that implement their own hostname verifier. Of those apps, 19,520 apps (98.9%) implement it in a way that accepts any hostname, i.e. they effectively turn off hostname verification and make their apps vulnerable to active Man-In-The-Middle attacks. Only the remaining 214 apps (0.1%) implement alternative hostname verification strategies. Although the limitations of static code anlaysis prevent us from assessing whether these implementations meet the programmers' expectations, we score them as secure compared to hostname verifiers that simply accept every hostname.

ICC Task. 42,193 apps implemented their own services. Of those, 15,857 (37.6%) configured a non-default access policy for their services by setting respective properties in the Manifest file. 11,929 (75.2%) of those apps use intent filters or set the `exported=true` flag, which weakens security. 3,928 (24.8%) of those apps configured their services to that an entity must have a permission in order to launch the service or bind to it. Only 101 apps required an entity to have a permission of the same developer or a system permission.

Secure Storage Task. 155,017 apps implemented file-system access. Of those, 34,183 (22.1%) access files on external storage or write to the internal storage with `MODE_WORLD_READABLE/WRITEABLE`. However, 120,834 (77.9%) only access files on internal storage with `MODE_PRIVATE`. Similar numbers can be seen with shared preferences, where 130,408 (88.0%) apps out of 148,256 use `MODE_PRIVATE` and 17,848 (12.0%) use a publicly accessible mode. SQLite databases are not very common among our dataset, but 4471 out of 4625 (96.7%) also use a private mode and only 154 (3.3%) a public mode.

Least Permissions Task. Overall we identified 64,232 apps that use intents to make phone calls. Of those apps 52,739 (82.1%) use the `ACTION_DIAL` action for that purpose. Interestingly 3,907 (7.4%) of those apps request the `CALL_PHONE` permission although `ACTION_DIAL` does not require a dedicated permission. The remaining 11,493 (17.9%) apps use the `ACTION_CALL` action which requires the `CALL_PHONE` permission to be requested by the developer. Of those apps, 6,157 (53.6%) do not request the `CALL_PHONE` permission and hence might crash if the `ACTION_CALL` intent is called.

C. Discussion

We found that 187,291 (93.6%) of the randomly chosen 200,000 apps we analyzed in our study used at least one of the APIs we used in our programming tasks, suggesting that our laboratory study includes programming tasks that real-world developers encounter. Interestingly, for the secure storage and least privilege tasks, most apps implement the more secure solutions. In contrast, for the secure networking and ICC tasks, we found more insecure solutions. This mirrors the results of our lab study (cf. Section V). This analysis provides additional concrete evidence for the relevance and the results of our lab study.

VIII. LIMITATIONS

As with most studies of this type, our work has several limitations.

First, the response rate for our online developer survey was very low, as might be expected from sending unsolicited emails to prospective participants. This may introduce some self-selection bias, but we have no reason to believe a priori that those who responded differ meaningfully in terms of security knowledge or resource usage from those who did not.

Our lab study created an artificial scenario—working within a tight time limit, with unfamiliar starter code—which may have impacted participants' ability to complete tasks correctly and securely. Similarly, the artificial nature of study participation may have reduced participants' incentives to consider security. In addition, a majority of our lab participants were students rather than professional developers, and overall the lab participants were more formally educated than the developers in our online survey, which may limit the generalizability of our results somewhat. The professionals in the study performed slightly but not significantly better than the non-professionals in functional correctness, but not in security. All of these issues, however, were present across conditions, suggesting that comparisons among conditions are valid. We also hoped that the time limit would partially emulate the

pressure professional developers feel to bring apps to market quickly rather than focus on writing the best possible software.

Our analysis of Stack Overflow threads is limited to only those accessed by our lab study participants; threads on other topics may exhibit different properties. In addition, our manual coding process was somewhat subjective. Nonetheless, we believe this analysis provides a useful glimpse into the broader characteristics of Stack Overflow as a resource.

The static code analysis we conducted has several limitations. Although we performed reachability analyses for all API calls, an inherent limitation of static code analysis is that we still might have included code paths that are not executed. For the ICC task, it is possible that some services we marked as insecure were made publicly available deliberately rather than by mistake; however, the official Android documentation [7] discourages the use of intent filters for security reasons. Hence, while we may have some false positives, our results do suggest at minimum a violation of best practices. A similar limitation applies to the storage task: while some uses of external storage are necessary or deliberate, this also represents a risky violation of best practices [8] that can lead to unexpected disclosures of personal information [35].

IX. Discussion

In the past, anecdotal evidence has suggested that the resources Android developers use when programming directly affect the security and privacy properties of the apps they make. In this paper, we present the first systematic investigation of this theory by approaching the problem of how programming resources affect Android developers' security- and privacy-relevant decisions from several different angles. We conducted a 295-person online survey about the resources developer use, both in general and specifically for security-relevant problems. Based on results from this survey, we then conducted a 54-person lab study directly exploring the impact of resource choice on both functional correctness and security. To provide context for these studies, we manually analyzed the security characteristics of the Stack Overflow posts our participants accessed and automatically analyzed how the APIs we tested in the lab are used in 200,000 randomly sampled apps from the Google Play market.

When combined, results from these varied analyses suggest several interesting conclusions:

- Real-world Android developers use Stack Overflow (and other Q&A communities) as a major resource for solving programming problems, including security- and privacy-relevant problems.
- Other resources, such as official Android API documentation, do not provide the same degree of quickly understandable, directly applicable assistance. Our results suggest that using Stack Overflow helps Android developers to arrive at functional solutions more quickly than with other resources.
- Participants who were given free choice of resources tended to visit both the official documentation and Stack Overflow, but their performance in both functional correctness and security was more similar to participants in the Stack Overflow condition.
- Because Stack Overflow contains many insecure answers, Android developers who rely on this resource are likely to create less secure code. Access to quick solutions via a Q&A community may also inhibit developers' security thinking or reduce their focus on security.
- Code relevant to the tasks we explored can be found in 93.6% of the apps we sampled. Many of these apps exhibit similar security patterns to those observed in our lab study.

Few participants in our study explicitly mentioned security or used it as a search term when accessing resources. While this may be partially a function of our artificial environment, when combined with prior research and anecdotal evidence, this suggests that security remains at best a secondary concern for many real-world developers [2], [18]. This underscores the need for both APIs and informational resources that promote security even when developers are not thinking about it directly.

Android developers are unlikely to give up using resources that help them quickly address their immediate problems. Therefore, it is critical to develop documentation and resources that combine the usefulness of forums like Stack Overflow with the security awareness of books or official API documents. One approach might involve rewriting API documents to be more usable, e.g. by adding secure and functional code examples. Another might be to develop a separate programming-answers site in which experts address popular questions, perhaps initially drawn from other forums, in a security-sensitive manner. Alternatively, Stack Overflow could add a mechanism for explicitly rating the security of provided answers and weighting those rated secure more heavily in search results and thread ordering. Further research is needed to develop and evaluate solutions to help prevent inexperienced or overwhelmed mobile developers from making critical mistakes that put their users at risk.

[7] Cf. http://developer.android.com/guide/components/intents-filters.html

[8] Cf. http://developer.android.com/guide/topics/data/data-storage.html

Acknowledgements

The authors would like to thank Sven Bugiel, Andrew Lui, and Yichen Qian for their support in the lab study, Marten Oltrogge for his contribution to the static analysis, Joseph Smith and Jennifer DeSimone for helping us navigate the IRB requirements for an international study, and all of the developers and/or students who kindly participated in our study. This work was supported in part by the German Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA), and by the U.S. Department of Commerce, National Institute for Standards and Technology, under Cooperative Agreement 70NANB15H330.

References

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. PScout: Analyzing the Android Permission Specification. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM, 2012.

[2] R. Balebako and L. F. Cranor. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security & Privacy*, 12(4):55–58, 2014.

[3] R. Balebako, A. Marsh, J. Lin, and J. Hong. The Privacy and Security Behaviors of Smartphone App Developers. In *Workshop on Usable Security (USEC'14)*, 2014.

[4] A. Baltadzhieva and G. Chrupala. Question Quality in Community Question Answering Forums: A Survey. *SIGKDD Explorations*, 17(1):8–13, 2015.

[5] A. Barua, S. W. Thomas, and A. E. Hassan. What Are Developers Talking About? An Analysis of Topics and Trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, 2012.

[6] B. Bazelli, A. Hindle, and E. Stroulia. On the Personality Traits of StackOverflow Users. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013.

[7] K. P. Burnham. Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods & Research*, 33(2):261–304, 2004.

[8] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague. OAuth Demystified for Mobile Application Developers. In *Proc. 21st ACM Conference on Computer and Communication Security (CCS'14)*. ACM, 2014.

[9] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing Inter-application Communication in Android. In *Proc. 9th International Conference on Mobile Systems, Applications, and Services (MobiSys'11)*. ACM, 2011.

[10] L. F. Cranor. A Framework for Reasoning About the Human in the Loop. In *Proc. 1st Conference on Usability, Psychology, and Security (UPSEC'08)*, 2008.

[11] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*. ACM, 2013.

[12] N. Elenkov. *Android Security Internals*. No Starch Press, 2015.

[13] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proc. 9th Usenix Symposium on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, 2010.

[14] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *Proc. 20th Usenix Security Symposium (SEC'11)*. USENIX Association, 2011.

[15] S. Fahl, S. Dechand, H. Perl, F. Fischer, J. Smrcek, and M. Smith. Hey, NSA: Stay Away from my Market! Future Proofing App Markets against Powerful Attackers. In *Proc. 21st ACM Conference on Computer and Communication Security (CCS'14)*. ACM, 2014.

[16] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM, Oct. 2012.

[17] S. Fahl, M. Harbach, M. Oltrogge, T. Muders, and M. Smith. Hey, You, Get Off of My Clipboard. In *Financial Cryptography and Data Security*, volume 7859, pages 144–161. Springer, 2013.

[18] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith. Rethinking SSL Development in an Appified World. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*. ACM, 2013.

[19] S. Garfinkel and H. R. Lipford. Usable security: History, themes, and challenges. *Synthesis Lectures on Information Security, Privacy, and Trust*, 5(2):1–124, 2014.

[20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. TThe Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM, 2012.

[21] S. Jain and J. Lindqvist. Should I Protect You? Understanding Developers' Behavior to Privacy-Preserving APIs. In *Workshop on Usable Security (USEC'14)*, 2014.

[22] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In *Proc. 2nd ACM CCS Workshop on Security and Privacy in Mobile Devices (SPSM'12)*. ACM, 2012.

[23] S. Komatineni and D. MacLean. *Pro Android 4*. Apress, 2012.

[24] P. G. Leon, L. F. Cranor, A. M. McDonald, and R. McGuire. Token Attempt: The Misrepresentation of Website Privacy Policies Through the Misuse of P3P Compact Policy Tokens. In *Proc. 9th Annual ACM Workshop on Privacy in the Electronic Society (WPES '10)*. ACM, 2010.

[25] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk. How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK. In *Proc. 22nd International Conference on Program Comprehension (ICPC '14)*. ACM, 2014.

[26] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang. Chex: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. In *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM, 2012.

[27] D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steenkiste, and C. Faloutsos. Analysis of the Reputation System and User Contributions on a Question Answering Website: StackOverflow. In *Proc. 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'13)*. IEEE, 2013.

[28] S. Nadi, S. Krüger, M. Mezini, and E. Bodden. "Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs? In *Proc. 37th IEEE International Conference on Software Engineering (ICSE'15)*, May 2016.

[29] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl. To Pin or Not to Pin—Helping App Developers Bullet Proof Their TLS Connections. In *Proc. 24th USENIX Security Symposium (SEC'15)*. USENIX Association, 2015.

[30] L. Onwuzurike and E. De Cristofaro. Danger is My Middle Name: Experimenting with SSL Vulnerabilities in Android Apps. In *Proc. 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec'15)*. ACM, 2015.

[31] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In *Proc. 21st Annual Network and Distributed System Security Symposium (NDSS'14)*. The Internet Society, 2014.

[32] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza. Understanding and Classifying the Quality of Technical Forum Questions. In *Proc. 14th International Conference on Quality Software (QSIC'14)*. IEEE, 2014.

[33] A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proc. 18th ACM Conference on Computer and Communication Security (CCS'11)*. ACM, 2011.

[34] A. Porter Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission Re-Delegation: Attacks and Defenses. In *Proc. 20th Usenix Security Symposium (SEC'11)*. USENIX Association, 2011.

[35] S. Son, D. Kim, and V. Shmatikov. What Mobile Ads Know About Mobile Users. In *Proc. 23rd Annual Network and Distributed System Security Symposium (NDSS'16)*. The Internet Society, 2016.

[36] D. Sounthiraraj, J. Sahs, G. Greenwood, Z. Lin, and L. Khan. SMV-Hunter: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps. In *Proc. 21st Annual Network and Distributed System Security Symposium (NDSS'14)*. The Internet Society, 2014.

[37] The Internet Society. Internet Society Global Internet Report 2015. http://www.internetsociety.org/globalinternetreport/assets/download/IS_web.pdf, 2015.

[38] C. Treude, O. Barzilay, and M. A. Storey. How Do Programmers Ask and Answer Questions on the Web? (NIER Track). In *Proc. 33rd International Conference on Software Engineering (ICSE'11)*, 2011.

[39] B. Vasilescu, A. Capiluppi, and A. Serebrenik. Gender, Representation and Online Participation: A Quantitative Study of StackOverflow. In *Proc. 2012 International Conference on Social Informatics (SocialInformatics)*. IEEE, 2012.

[40] S. Wang, D. Lo, and L. Jiang. An Empirical Study on Developer Interactions in StackOverflow. In *Proc. 28th Annual ACM Symposium on Applied Computing (SAC'13)*. ACM, 2013.

[41] W. Wang and M. W. Godfrey. Detecting API Usage Obstacles: A Study of iOS and Android Developer Questions. In *Proc. 10th Working Conference on Mining Software Repositories (MSR'13)*. IEEE, 2013.

[42] W. Wang, H. Malik, and M. W. Godfrey. Recommending posts concerning api issues in developer q&a sites. In *Proc. 12th Working Conference on Mining Software Repositories (MSR '15)*. IEEE, 2015.

[43] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission Evolution in the Android Ecosystem. In *Proc. 28th Annual Computer Security Applications Conference (ACSAC'12)*. ACM, 2012.

[44] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang. The Impact of Vendor Customizations on Android Security. In *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*. ACM, 2013.

[45] Y. Xie and D. Powers. *Statistical Methods for Categorical Data Analysis*. Emerald, 2008.

[46] Y. Zhou and X. Jiang. Detecting Passive Content Leaks and Pollution in Android Applications. In *Proc. 20th Annual Network and Distributed System Security Symposium (NDSS'13)*. The Internet Society, 2013.