From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security

Charles Weir, *Lancaster University;* Ben Hermann, *Paderborn University;*
Sascha Fahl, *Leibniz University Hannover*

https://www.usenix.org/conference/usenixsecurity20/presentation/weir

This paper is included in the Proceedings of the
29th USENIX Security Symposium.

August 12–14, 2020

978-1-939133-17-5

From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security

Charles Weir, *Lancaster University*
Ben Hermann, *Paderborn University* Sascha Fahl, *Leibniz University Hannover*

Abstract

Increasingly mobile device users are being hurt by security or privacy issues with the apps they use. App developers can help prevent this; inexpensive security assurance techniques to do so are now well established, but do developers use them? And if they do so, is that reflected in more secure apps? From a survey of 335 successful app developers, we conclude that less than a quarter of such professionals have access to security experts; that less than a third use assurance techniques regularly; and that few have made more than cosmetic changes as a result of the European GDPR legislation. Reassuringly, we found that app developers tend to use more assurance techniques and make more frequent security updates when (1) they see more need for security, and (2) there is security expert or champion involvement.

In a second phase we downloaded the apps corresponding to each completed survey and analyzed them for SSL issues, cryptographic API misuse and privacy leaks, finding only one fifth defect-free as far as our tools could detect. We found that having security experts or champions involved led to more cryptographic API issues, probably because of greater cryptography usage; but that measured defect counts did not relate to the need for security, nor to the use of assurance techniques.

This offers two major opportunities for research: to further improve the detection of security issues in app binaries; and to support increasing the use of assurance techniques in the app developer community.

1. Introduction

Increasingly software security and privacy are becoming major problems for society. Almost every day we hear of new attacks and privacy problems, and increasingly they are affecting not just large companies, but everyone [46]. While there are many ways to address these issues, clearly software developers have a vital role to play in creating services and applications that enforce security effectively[1].

The software industry has developed a range of inexpensive security assurance techniques for software developers [9,45,51] and some teams even use formal secure development lifecycles to pull them together [55]. However, though many developers are using those assurance techniques, others are not. Factors such as lack of motivation, pressures to do other work, lack of access to learning and support, or sheer ignorance of the need, all act as barriers to adoption [5,32]. Some development teams may have access to security experts to help them; others may have little or no practical knowledge of software security. In some cases, this may not matter—if the code has no security or privacy implications—but in others it may harm a range of stakeholders, from software users to organization senior management.

In this work we investigate how big a problem this may be in practice. Our first research question was:

RQ1: *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in the development team?*

To begin to address this question[2], we chose a specific set of software developers to investigate: Android application developers. Our reasons for choosing these were twofold:

1. The research team has considerable experience in Android development security research [2,33]
2. The Android ecosystem provides access to both developers and the software developed, along with an indication of application usage.

Accordingly, we carried out an online survey of professional Android developers, asking for details of their security practices and interactions. Our key findings from statistical analysis of the 330 completed and accepted surveys[3] are as follows:

- No more than 22% of Android app developers have regular access to security professionals;

[1] Throughout this paper we refer to ’developers’ meaning all those involved with software development: programmers, testers, project managers, and product owners.

[2] RQ1 was modified to include ‘how’ and ‘perceived’ following feedback on the paper.

[3] Assuming the sample is representative of Android app developers. See Section 5.1.

- Less than 53% of them have used any of the basic assurance techniques; less than 30% use any regularly; and security updates for apps generally happen less than once a year.
- Less than 15% of them have made more than cosmetic changes as a result of the new GDPR legislation.
- Android app developers' use of assurance techniques is positively correlated with the perceived need for security, the involvement of security experts or champions, and the security expertise of the developers;
- The reported frequency of app security updates is positively correlated with the perceived need for security, the security expertise of the developers, and the developers' use of assurance techniques.

In a second phase, we investigated how these aspects of the development process were reflected in objective app security outcomes. Our research question for this phase was:

RQ2: *To what extent do the need for security, the involvement of specialist roles, and the use of assurance techniques in a development team lead to fewer security defects?*

We analyzed the corresponding Android applications created by each developer and matched the findings to the questionnaire results, concluding that:

- There was no correlation found between the perceived need for app security, nor the use of assurance techniques, and the defect count of the resulting app; and
- Surprisingly, the involvement of security professionals and 'security champions' is correlated with higher cryptographic API defect counts.

This paper is structured as follows. Section 2 explores related work, including a discussion of assurance techniques; Section 3 describes the survey design, participant recruitment approach, analysis plan, survey trials and limitations; Section 4 describes the same for the app binary analysis; Section 5 explores both the survey and app analysis results; Section 6 explores the implications of these results; and Section 7 summarizes the main learning points and conclusions.

2. Related Work

In this section, we discuss related work in three key areas: ways of finding security and privacy flaws in otherwise benign mobile apps; research work into developers' secure development behavior; and findings on the important developer assurance techniques.

2.1. Security and Privacy in Mobile Apps

The introduction of App Stores, that act as an intermediary between developers and consumers, has required each app store provider to find ways to detect rogue applications and rogue application developers. This has led to research into ways of analyzing application binaries to detect hostile behavior. Enck et al. [18], for example, used a decompiler to analyze a range of popular applications, finding many privacy issues though no security misbehavior. Glanz et al. [22] inspected obfuscated apps to detect repackaged apps—benign apps that have been modified and re-uploaded to app stores. Reyes et al. [39] explored children's app binaries, finding many violations of US privacy law.

However, only more recently has there been much investigation into the problems of benign apps that may have security or privacy flaws. This may be due partly the difficulty of taking action: Google Play does not have the remit of enforcing better security [29] and the app developers may not wish to do so. But with the increase of interest in security issues [46], researchers are now taking a variety of approaches to investigate.

Li et al. [28] provide a literature survey over the vast amount of research in the field of static program analysis for Android including an overview of used tooling and methodology. The most prominent works in the area are FlowDroid by Arzt et al. [4], which is able to find privacy leaks by inspecting illicit information flow; IccTA by Li et al. [27], which extends FlowDroid to account for inter-component privacy leaks; and MalloDroid by Fahl et al. [20], which detects improper use of transport layer security in apps.

As Android apps become increasingly polyglot with the use of hybrid app frameworks and native libraries, in recent work, analyses over these language boundaries have been increasingly in focus. Bai et al. [7] inspected apps which use the JavaScript bridge communication scheme to construct leaks undetectable by previous approaches. Wei et al. [50] provide support for information leak tracking through the Java and the native part of an app helping to find information leakage with could not be detected with Java-only-based approaches.

Another important area of investigation is the security of the interaction of apps with cloud environments. Zuo et al. [58], for example, found by inspecting apps from Google Play that many of the used cloud services are vulnerable and may leak user data—an observation previously made by Rasthofer et al. [38].

2.2. Developer Security Behavior

A few teams have investigated the underlying causes behind software security problems. Oliveira et al. [32] used psychological manipulation to explore what caused developer volunteers to include vulnerabilities in software, finding two main causes: developers' focus on 'normal cases' and a lack of priority for security. Assal and Chiasson [5] surveyed 123 North American developers, finding their respondents motivated to produce secure code—once the implications and possible damage to stakeholders are understood—but often prevented by lack of organizational and process support.

Senarath and Arachchilage [42] used a task given to programmers to explore issues related to user privacy; their findings were that it was difficult to understand such requirements and to translate them into engineering techniques.

Others have investigated the use and adoption of security–focused code analysis tools. Xie et al. [57] explored the impact of one such tool, finding that even when creating secure code is relatively easy developers still need motivation to make the needed changes. Witschey et al. [56] surveyed developers about their adoption of such tools, finding that the most important factor was seeing peers using them.

Several researchers have investigated the process of updating software when security faults are detected. Derr et al. [15] investigated how Android app developers keep library versions up to date, surveying app developers and analyzing of app binaries. They found that it was often possible to solve vulnerabilities by library updating without changes in code, but that frequent backward incompatible changes and incorrect Semantic Versioning in libraries currently make such updates difficult. Others investigated to what extent the fixes were necessary: Nayak et al. [30] found that less than 15% of known vulnerabilities were actually used in attacks, suggesting an opportunity for a more nuanced approach than just fixing everything. Vaniea and Rashidi [49] used a survey of 307 users to analyze the effectiveness of the update procedure. They derived advice for developers, including making it easy to find documentation, and planning a 'recovery path'.

Other researchers have investigated security requirements, especially related to privacy. Türpe [47] found a range of research related to security requirements, especially Threat Modeling techniques, but no agreement on terminology or approach.

2.3. Developer Assurance Techniques

An important approach to improving software quality has been changes to development processes. This may be through a *Secure Software Development Lifecycle*, a prescriptive set of instructions to managers, developers and stakeholders on how to add security activities to the development process [55]; or by empowering the developers to make their own decisions about how to achieve security [53].

Particularly important is the need to align security goals with business needs [10,51]. Though much work has been done to support evaluating security problems in terms of risk and impact [47], identifying the need for security experts to be business negotiators and evangelists [23], there has been little attention to developer interactions with other stakeholders on security.

The specific techniques and approaches used by developers depend, of course, on their environment and constraints. There are more than twenty identifiable assurance techniques in regular use today, differing significantly in cost effectiveness, though there are combinations that are typically used together [45]. In particular one can identify a set of about five 'entry level' assurance techniques that are widely used and can be introduced at relatively low cost [51]. In terms of practical support for developers, a recent book 'Agile Application Security' by Bell et al. [9] provides guidance, a discussion of tools and detail on a range of assurance techniques.

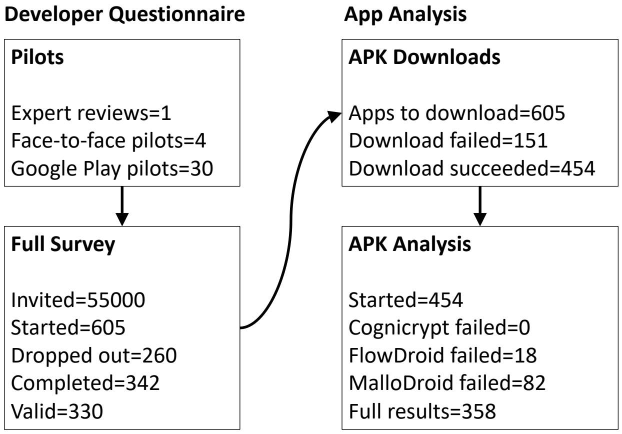

Figure 1: Study Procedure

2.4. Related Work Summary

Though there has been considerable work done on identifying practical assurance techniques and tools for security, and some work on motivating developers to use them and investigating reasons for vulnerabilities, there has been little or no work investigating whether the need for security does in practice correlate with better practices, and result in better security.

In this paper we make a start at that investigation.

3. Survey Methodology

We conducted an online survey of Google Play Android developers in May 2019, receiving 345 complete responses. This section provides a detailed overview of our methodology, with the goal of making our research plan both transparent and reproducible, to allow readers and future researchers to better assess our contribution. Figure 1 summarizes the study procedure.

3.1. Survey Questionnaire Structure

We asked our respondents to answer questions about their Android application development behavior and context relevant for application security and privacy, and a set of demographic questions. Although this might have led to self-reporting and social desirability bias, we considered this approach the best practical approach to address the research. We implemented the questionnaire in Qualtrics [37], and developed it using an iterative process.

Appendix B contains the full list of questions. In summary, we asked respondents:

- Whether they worked in a team, and if so their role and the team size;
- The Android development environments they used;
- The number of recent releases for their most frequently updated app, and the proportions of updates addressing each of new features, library updates, security and privacy issues;
- Their evaluations of the importance of security and of privacy, both implicitly and for sales;
- Whether they receive support from security professionals or internal security champions, and if so, the nature of that support;
- What events had led to recent changes in security;
- Which secure development practices they used, and to what extent;
- How long they had been programming, both generally and with Android;
- How many apps they had developed, and whether it was their primary job; and
- Demographic information about gender, language, and country of residence.

Definitions: In the questions, 'recent' was defined as the previous two years, and 'security champion' to be a non-expert who takes a particular interest in security [8]. We asked developers with more than one app to provide answers for the most frequently updated one.

Secure Development Practices: The questions about secure development practices asked specifically about five of the most frequently-used assurance techniques [45,51], as follows:

Threat Assessment	Working as a team to identify actors and potential threats; following this up with risk assessment and mitigation decisions.
Configuration Review	Keeping components up-to-date using component security analysis tools to the toolchain.
Automated Static Analysis	Using code analysis tools to identify certain categories of security vulnerability.
Code Review	Having other programmers or security experts review code for security problems.
Penetration Testing	Having external specialist security testers identify flaws.

Question Wording: All the questions about security processes were worded as questions of fact, rather than of future intentions as in some security surveys [16], to reduce the impact of desirability biases.

Omissions: We considered asking about code analysis tools, since these are of particular interest to researchers. However, static analysis is only one of the five assurance techniques considered, and investigating all the techniques would have made the questionnaire unacceptably long without contributing to answers for the research questions.

3.2. Survey Pre-Testing

After developing an initial questionnaire, we conducted a set of pre-tests to glean insights into how survey respondents might interpret and answer questions, and how long they might take to complete the survey, as follows.

Expert Review: After developing and revising a first version of the survey questionnaire, we asked an experienced usable security and privacy researcher with survey expertise, who is not part of the research team, to review our survey questionnaire and evaluate question wording, ordering, and bias. Expert reviewing is a method that supports identifying questions that require clarification and uncovering problems with question ordering or potential biases [36]. Following the expert review, we improved the wording of several questions, and changed the survey software configuration to randomize the order of answers and questions wherever this was possible.

Face-to-face Testing: To test our survey questions under realistic conditions, we then identified four local Android developers who were not previously involved in the research project, and asked each to complete the survey while discussing it with a researcher. As a result, we modified the wording of two questions and added one. We also noted that responses from those who had produced only simple apps were not interesting from a security viewpoint, and accordingly modified our criteria for invitations to only invite developers of 'successful' and 'maintained' apps: ones that had received more than 100 downloads and at least one update.

Pilot Survey: To further test the questionnaire, we ran a set of pilot surveys with Android developers drawn from the same invitation list as the main survey (Section 3.4), inviting 5000 and gaining 30 completed entries. Participants of the pilot were excluded from the full survey.

We used the results to check that the number of drop-outs during the survey was acceptable; it was, since of those who completed the first page of questions, only 21% dropped out later in the survey. In the pilot questionnaire we used a text field for developers to answer what changes they had made as a result of GDPR; we coded the pilot responses, and provided the most frequent answers as 'tick boxes' in the final survey.

The results also helped focus and plan our analysis of the data.

Specifically, we identified the following additional research questions to help scope the problem of supporting developers:

RQ3 *What proportion of Android developers have access to security experts, and*
RQ4 *To what extent do Android developers actually use assurance techniques?*

3.3. Calculation of Required Sample Size

We used Fowler's guidance [21], identifying the smallest subgroups for which we wanted data, using the pilot data to estimate the proportion of these, and making the sample size large enough to get significant data from these groups. The key subgroups were those developers working with security professionals, and those using assurance techniques; and we chose to get between 50 and 100 in each group to give typical sampling errors on data for each subgroup of between 4% and 15%. Based on the pilot data, therefore, we calculated a target sample size of 310, requiring us to send 55,000 invitations.

3.4. Recruitment

We invited only registered Google Play developers. From January to February 2019 we crawled the details' pages of 3,608,673 (2,087,829 free and 1,520,844 paid) Android applications from those published in Google Play. For all apps, we stored their last update time, name, developer data and download counts.

Overall, we identified 312,369 developer accounts that match the 100+ downloads and update requirements in Google Play. The number of apps published by a single developer account in that sample ranges from 1 to 3,302 with a median of 2. From these 312,369 developer accounts, we selected a random sample of 55,000, and sent a single invitation email to each to ask them kindly to support our research. Of the invited 55,000 participants, 605 started and 345 completed the survey. Ten of the invited developers reached out to us via email. None complained about being contacted; three asked to be removed from the mailing list; the remainder provided various reasons for not completing the survey, including two who noted the security questions and stated that their apps had no security aspects. 240 took the opportunity to leave their email address in the survey questionnaire for us to send them the results of this work.

3.5. Filtering Invalid Results

In psychological surveys, a common stratagem is to ask a question twice, once negated. One can then filter out meaningless responses (or use them to calculate a "self-consistency" score for the survey). Since our survey was asking facts rather than personality, we concluded that this would be contrived and irritating to the respondents. Instead we looked at response times, experimented to find a minimum time that a participant might be expected to take to complete the survey, and filtered out the few (10) surveys that had taken less than that minimum time to complete.

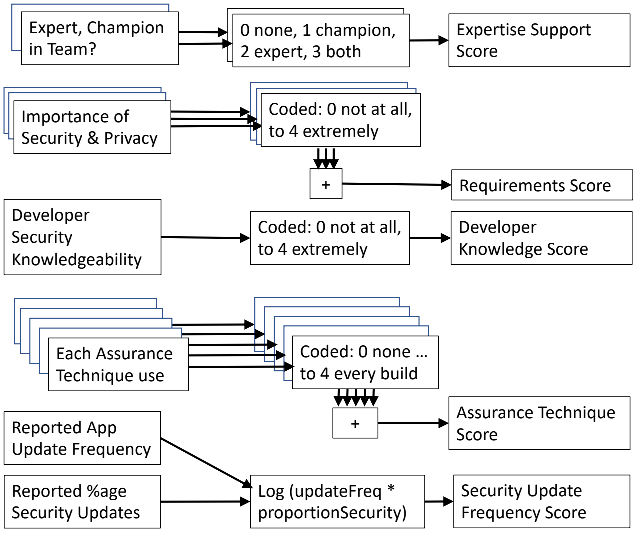
Expert, Champion in Team?
0 none, 1 champion, 2 expert, 3 both
Expertise Support Score
Importance of Security & Privacy
Coded: 0 not at all, to 4 extremely
+
Requirements Score
Developer Security Knowledgeability
Coded: 0 not at all, to 4 extremely
Developer Knowledge Score
Each Assurance Technique use
Coded: 0 none … to 4 every build
+
Assurance Technique Score
Reported App Update Frequency
Reported %age Security Updates
Log (updateFreq * proportionSecurity)
Security Update Frequency Score

Figure 2: Survey Security Scores

3.6. Survey Statistical Analysis Plan

This paper uses four forms of statistical analysis:

1. Population analysis, to explore how well our sample corresponds to the larger population;
2. Graphical analysis, to show the nature of the data;
3. Confidence limits for proportions in the wider population based on proportions in the sample; and
4. Correlation analysis, to identify relationships between different data items.

We defined the statistics scores and outline analysis methods before collecting the main survey data, as required for research best practice [11,12]. For analysis, we used Python statistical packages, including Pandas, Statsmodels, and Seaborn, in Jupyter Notebooks [25].

Linear Analysis for RQ1: To address RQ1, we defined scores based on each respondent's survey answers: some scores captured the "*need for security and privacy*" (the independent, 'input', variables); others the "*security-enhancing activities and interactions in the development team*" (the dependent, 'output', variables).

Figure 2 shows the processing we did to create these scores. The aim in each case was to create an ordinal score that approximated to linear across the range of raw data, so a higher score corresponds to more security (or more drivers towards security) and each increment represents a similar semantic increase. As shown, the Requirements Score reflects the *security need* as the arithmetic sum of the three Likert-style responses encoded as integers; similarly, to explore the *why*, there are Developer Knowledge and Expertise Support scores. We estimated a Security Update Frequency as the product of the answers to two questions; this had an exponential (Poisson) distribution, so to make it linear [3] we used a transformation: $\log(x_i + 1)$ to create the Security Update Frequency Score. Appendix C provides more details.

The calculation of the Expertise Support Score is based on an assumption that direct expert involvement is more effective than 'security champions'; the Requirements Score assumes that, for example, occasionally using two techniques is as effective as regularly using one; and the Assurance Technique Score assumes that, say, considering four techniques is as effective as consistently using one. Though reasonable as an approach, none of these scores are linear or even provably ordinal [44]; we anticipated that inconsistencies in the scoring would add to the statistical variance but not obscure overall trends. See Section 5.5 for a post-hoc justification.

In statistics, the usual relationship to look for is a linear one. In line with previous research in the field [16] we used the Pearson Correlation Coefficient ('Pearson R') calculation [14] to establish whether pairs of values had a significant linear relationship; this test is acceptable for Likert-style data [24,31].

Given that the scores were not provably linear, we also investigated a more sophisticated modelling technique, creating Decision Tree models [41] for pairs of scores and using F-Tests [13] to compare each with the simpler Pearson R model.

In this analysis we treated the Security Update Frequency score as a dependent variable (output); and the Requirements, Expertise Support, and Developer Knowledge scores as independent variables (inputs)[4]. The use of Assurance Techniques is likely to be affected by the latter three variables but may itself in turn affect the Security Update Frequency and other security outcomes; in the analysis, therefore, we treated the Assurance Technique score both as an independent and as a dependent variable.

Since the analysis constituted multiple tests on the same data, we applied the Bonferroni correction [40], reducing the threshold for 'significance' accordingly to $(5\%)/5 = 1\%$. To validate the preconditions for the Pearson Correlation Coefficient test [14], we then constructed x-y plots of all the pairs of variables that showed significant correlation.

4. Application Analysis Methodology

In the second phase of the project, we downloaded and analyzed the apps corresponding to the survey responses. For analysis, we used a selection of state-of-the-art of vulnerability scanners. Each one focuses on a different problem category and produces a relatively low number of false positives. We chose mature tools that are openly accessible to Android developers.

4.1. Description of Analysis Tools

The tools covered three key areas: SSL Security, Cryptographic API Misuse, and Privacy Leaks. We selected these areas based on previous work and because these cover a representative range from the possible security and privacy vulnerabilities faced by application developers [34].

SSL Security: A key concern in the secure treatment of information is the correct use of secure transport mechanisms (SSL, TLS) when connecting to remote systems. To capture this aspect, we used two techniques. First, we used MalloDroid [20] to inspect the correct use of certificate validation in the apps code. Second, we extracted any HTTPS URLs from the constant pools of the classes contained in the app using the `OPAL` framework [17] and checked the corresponding server configurations and certificates using the command-line tools `curl` and `openssl`.

Cryptographic API Misuse: Many apps use cryptographic measures to improve data security and privacy, and a key concern in the secure treatment of information is the handling of cryptographic primitives (e.g., for persistence). We run CogniCrypt [26] to capture this aspect. CogniCrypt uses static inter-procedural static program analysis to detect misuses of the Java Cryptography API. The detected problems range from improper configuration of algorithms (e.g., use of AES with ECB) to incorrect order of calls to the API. As it is formulated as a static program analysis, CogniCrypt makes conservative assumptions (over-approximations) on the control flow of the program, which may produce false positive reports.

Privacy Leaks: To find possibly harmful data flow that can lead to privacy leaks, we used FlowDroid [4]. This tool is designed to find information flow in Android apps between defined information sources and information sinks. For example, the location APIs are considered as sources of private information, and the text message sending APIs as sinks. FlowDroid uses static inter-procedural data flow analysis to find evidence of directed information flow between these methods. We configured the tool with the default sources and sink for Android provided by the authors, which had been constructed by manual inspection of common vulnerabilities in Android apps. FlowDroid is not able to determine if the found information flow is to be considered an actual leak as it might also be intended to use the information in the particular context (e.g. for location-based services).

Practical Approach: We downloaded the application binaries for at least one application by each of the survey respondents,

[4] Pearson's R does not distinguish dependent and independent variables, so this affects only our choice of scores to correlate with each other.

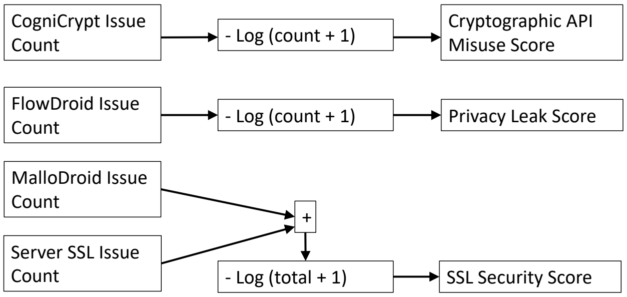

Figure 3: App Analysis Security Scores

wherever possible; we ran the full set of scanning tools on each, and counted the *issues* (reports of possible vulnerabilities) generated. Appendix A lists the versions of the tools we used.

4.2. Application Statistical Analysis

As in the previous phase, we used graphical tools to explore the data, and linear analysis to explore relationships between the data.

To investigate RQ2, we defined further scores to represent the outcome *"fewer security defects"* in each app analyzed. Figure 3 shows the processing involved. We anticipated that the issue counts would have a Poisson distribution; to permit linear analysis we used a log transformation[5]. As with the scores for developer behavior, we wanted scores that increase with increasing app security and privacy, and we therefore negated the log value.

We used the same method as previously (Section 3.6) to look for relationships between these scores and the scores from Figure 2 covering the "*need for security, involvement of specialist roles, and use of assurance techniques in a development team"* in RQ2.

4.3. Ethical Considerations

Our institutions' Institutional Review Boards approved this study, including the use of publicly available contact details for the survey invitations. Additionally, we modeled our research plan and survey procedure to adhere to the strict data and privacy protection laws in the UK and Germany and the General Data Protection Regulation in the E.U. We provided all participants with a form that informed them about the study purpose, the data we collected and stored, and an email address and phone number to contact the principal investigators in case they had questions or concerns.

4.4. Survey Limitations

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

The response rate for our online developer survey was very low, as might be expected from sending unsolicited emails to prospective participants. However, our recruitment approach was incorporated by relevant previous work [1,2,54]. The low response rate may introduce some self-selection bias, but since the invitations made no mention of security, we have no reason to believe a priori that those who responded differ meaningfully in terms of security or privacy behavior from those who did not.

All the survey data—except download count and last app update date—is self-reported. Though we addressed this by keeping questions as fact-oriented as possible, this is an important limitation.

In terms of the population, the survey reached app owners rather than all app developers; so, data about the respondents' own experience is not representative of all Android developers, nor of software developers in general.

4.5. App Analysis Limitations

The static analyses we chose each consider specific categories of vulnerabilities. This may disregard other categories of issues which may also be security critical. Indeed, many vulnerabilities—especially privacy ones—will tend to be in the intended app functionality rather than in the detailed implementation, and we have no way to estimate these. However, we used detectors for a range of implementation issues which may be found through other methods, and which developers who consider security or privacy important would be expected to address.

Static program analysis tools often report false positives, and the tools we used are no exception. Our approach for this survey, however, was to assume that the reported issue counts will correlate with the numbers of true vulnerabilities, and therefore that such counts can be used as a proxy for aspects of app security in statistical analysis.

We were able only to analyze 'free' and 'freemium' apps, not ones where Google Play Store charges for download; this may introduce a bias. In cases where respondents have more than one app, the app we downloaded may not be one requiring the security practices and priorities described in the survey.

We considered improving the app analysis by ranking vulnerabilities based on severity. However, the analysis did not identify vulnerabilities; it reported counts of 'issues' detected, where an 'issue' is a potential vulnerability. To determine whether an issue represents a vulnerability would require detailed analysis of the source code; this source code was not available to the researchers, and decompilation was infeasible due to the widespread use of obfuscation tools.

[5] Specifically, $\log(x_i + \mathrm{k})$, where k is chosen to minimize skewness [3]; in practice we trialed different values of k, finding no difference to the results, so used the conventional research practice of k=1.

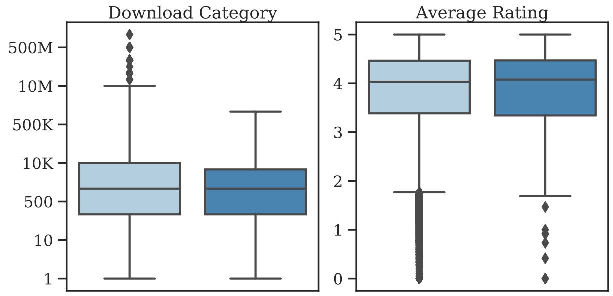

Figure 4: Comparing Invitees (light blue) with Respondents (dark blue)

We also considered distinguishing issues in the source code from issues in libraries, or using vulnerability ratings for libraries. However, although there have been several worthwhile tools developed to analyze the libraries used by Android apps, including LibScout [6] and LibDetect [22], with the current state of the art they are not sophisticated enough to detect library versions reliably, nor are they integrated with other binary analysis tools to allow differentiation of issues in libraries from issues in the main code.

5. Results

This section describes our results, both from the survey and from the app analysis.

5.1. Sample Validity

Comparing the box plots for invitees with those for participants in Figure 4, we see that the average user rating and number of downloads for apps produced by the 345 developers who completed surveys are very similar to those for the 55,000 invited.

One survey question asked the respondent's years of experience in software development. Figure 5 compares the results with answers to a similar question addressed to the 21,000 Android developers out of the 89,000 developers who answered the 2019 Stack Overflow developer survey [43]. As will be seen, the respondents are generally more experienced than the corresponding general population (median 12 years; population median of 8 years; Mann Whitney $p = 10^{-21}$).

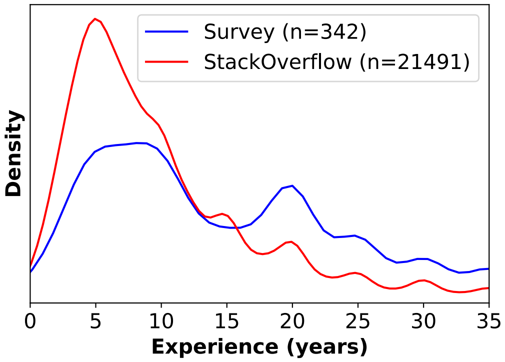

Figure 5: Development Experience

One concern was whether our app selection criterion (over 100 downloads and one update) was too lenient, since little-used apps may well have poor security. To test this, we used the Mann Whitney test comparing developers of apps with less than 1000 downloads against the rest[6]. We did this for all of the scores (Sections 3.6 and 4.2) and for all the numerically analyzable survey questions to see if the distribution was different for low-download apps. In the survey results and scores we found small p-values (<0.003) only for questions whose answers we expected to correlate with download counts: 'How many apps have you developed', 'How many Android apps have your developed' and 'Is developing apps your primary job', and we concluded that the populations were essentially the same. Doing the same Mann Whitney test on the scores in Phase 2, we found low p-values only for the Cryptographic API Misuse and Privacy Leak scores (p ~ 0.016 for each). Though suggestive, these values are not statistically significant given the number of tests done on that same data. We concluded that there was no justification for changing our app selection criteria.

Finally, to check the accuracy of respondents' replies, we compared the respondent-stated app update interval with objective evidence. App update histories are not generally available from Google Play, but we did collect the last update date for each app we considered. We correlated the time since that last update with the participant-stated update interval using log scales: Pearson R=0.38, P=1e-9 (n=242). The tiny P value corroborates the assumption that the stated update frequencies reflect reality; the moderate R value reflects that respondents were asked the about updates to 'their most frequently updated app' and not the app we considered, plus the randomness of where each app was in the release cycle.

5.2. Findings on Self-Reported Developer Behavior

The next sections describe the survey results for individual survey questions, without considering associations between answers[7].

Importance of Security and Privacy: Figure 6 shows respondents' ratings of the importance of security and privacy in their apps. For comparison, we also asked and show the importance of other functional and non-functional requirements. We were surprised how many developers considered security and privacy important, with ratings comparable with multi-platform support and higher than for many features.

[6] We specified this analysis after data gathering; accordingly, significance in any of the correlations should be considered suspect. However, a lack of significance in a wide range of correlation calculations is a valid finding.

[7] The number of answers varies to each question or set of questions, giving different values for 'n' in each chart.

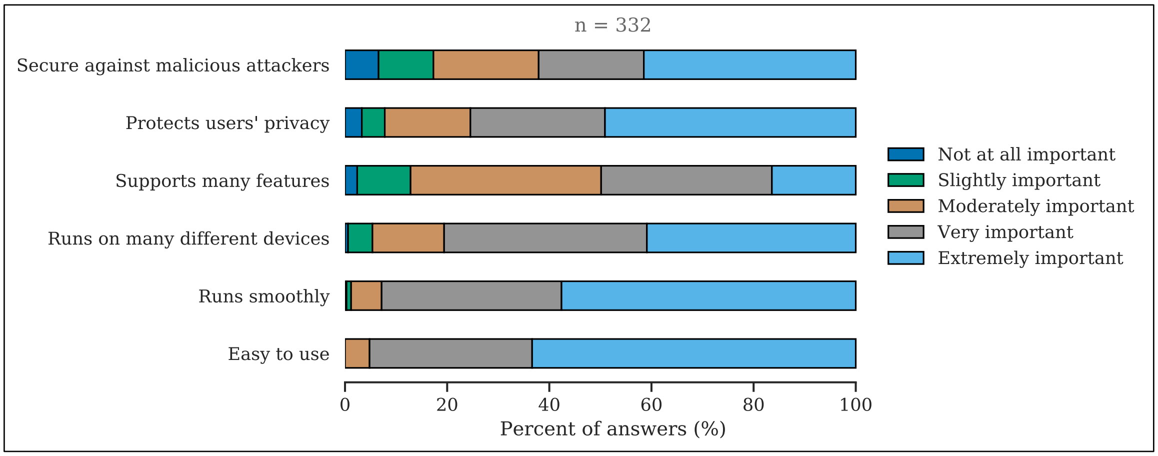

Figure 6: Importance of Different Requirements

Team Structure: Only 42% of respondents were working in teams, the remainder being solo developers. Only 17% of respondents received support from professional security experts. So, for RQ3 we calculate the ninety-five percent confidence interval [48] for the proportion working with security experts in the Android app developer population as a whole as:

Lower bound = 14%, Upper bound = 22%

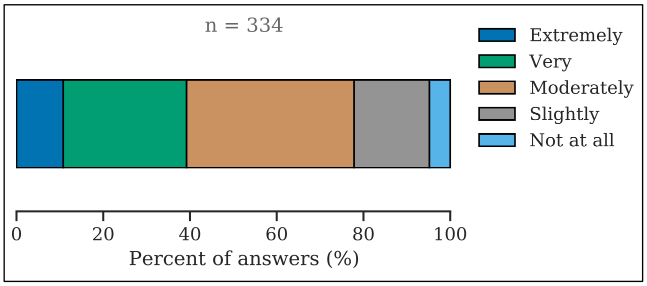

Figure 7: How Knowledgeable about Security

Of these few professional security experts discussed by respondents, 33% were part of the development team and the remainder external. Their most common function was Penetration Testing (44%), but they also provided Design Reviews (39%), Audits (33%) and Training (27%).

Some teams (18%) had a ‘security champion’, a non-expert providing security input to the rest of the team. Only 7% had both professional experts and champions.

Developer Security Knowledge: Figure 7 shows how survey participants rated their security expertise. Interestingly, very few considered themselves to have no knowledge; this is as we would expect given the level of development experience of participants (Section 5.1).

Use of Assurance Techniques: Figure 8 shows the reported use of assurance techniques. Unsurprisingly, Threat Assessment for every build is rare (possibly those respondents consider the list of threats every day), as is Penetration Testing (automated penetration testing, perhaps; one participant explicitly mentioned doing this). But otherwise the proportions using each are fairly consistent across all the techniques.

Combinations of Assurance Techniques: We investigated the extent to which teams used combinations of assurance techniques. Figure 9 summarizes how many and how often the techniques are used. It shows the proportion of respondents using each number of the techniques (at least), separated out to show how often they used them. As will be seen, less than half had used even one technique; about a quarter used one or more regularly; and very few used as many as four regularly.

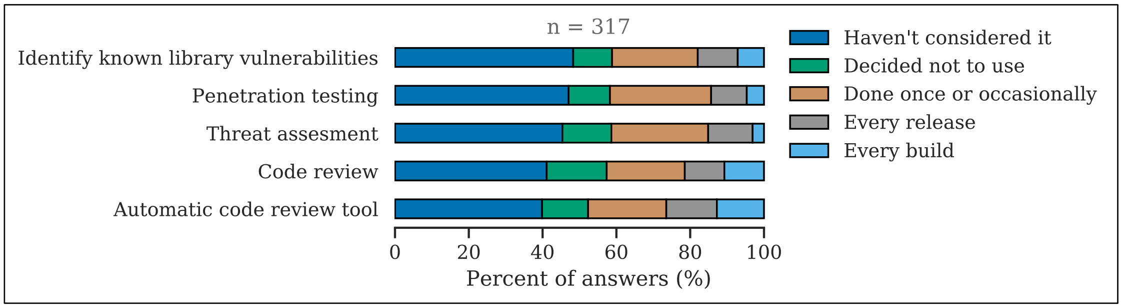

Figure 8: Use of Assurance Techniques

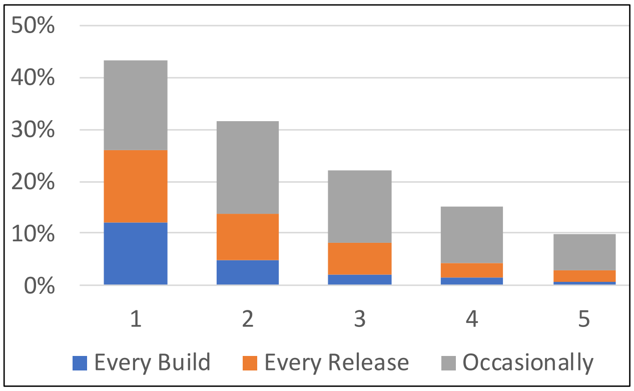

Figure 9: Proportion Using N Assurance Techniques

So for RQ4, the 95% confidence intervals for the proportion regularly using one or more of the given assurance techniques in the wider Android developer population [48] are:

Lower bound = 22%, Upper bound = 30%

We analyzed which combinations of techniques were popular amongst the 14% (57) of respondents who only used two or three regularly. The most popular were:

Auto. Static Analysis	Config. Review	37%
Auto. Static Analysis	Code Review	32%
Code Review	Config. Review	21%
Threat Modelling	Penetration Test	18%

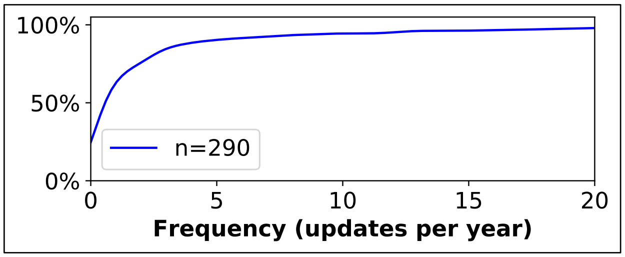

Figure 10: Security Update Frequency (Cumulative)

Security Updates: Figure 10 shows the frequency of security updates, calculated as the product of the reported update frequency, and the reported proportion of security updates. The 95% confidence interval for the proportion with less than one update a year is 59% - 70%.

5.3. Recent Changes in Team or Development Security

Given how fast moving the field of software security has become, it is also important to know what might have caused changes in the developers' perceptions or actions around security. Two questions in the survey addressed this: one listing possible reasons for security and privacy improvements and asking which had affected app security; and for those who mentioned an impact from the recent European GDPR legislation [19], a further question asking what changes they had made as a result. Since the GDPR legislation affects any apps sold in Europe, it impacts developers worldwide.

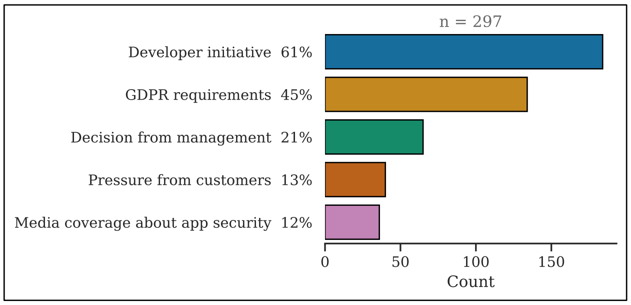

Figure 11: Top 5 Reasons for Security Changes

Figure 11 shows the answers. Interestingly, the developers' perception is that, even more than GDPR, the main security driver has been the developers themselves. Encouragingly very few (3%) reported security improvements as a consequence of actual security issues affecting themselves, suggesting that this is still rare; a few more (7%) reported 'horror stories'—something bad happening to a competitor.

Of the 45% of participants (n=133) who reported changes as a result of GDPR, Figure 12 summarizes the changes they made as a result. We observe that the majority of these changes were cosmetic, at least as far as the app itself was concerned: changing privacy policies or adding pop-up dialogs. Only 33 made substantive changes to improve user security or privacy (giving 95% confidence limits of 8% to 15% for the wider Android developer population [48]).

5.4. Linear Analysis of Developer Survey Scores

Table 1 shows the results of the analysis described in Section 3.6. It correlates each of the two dependent scores representing "security-enhancing activities and interactions in the development team" against four independent "need and mechanisms for security and privacy" scores. Non-italic figures highlighted in yellow indicate a statistically significant result (p<0.01)

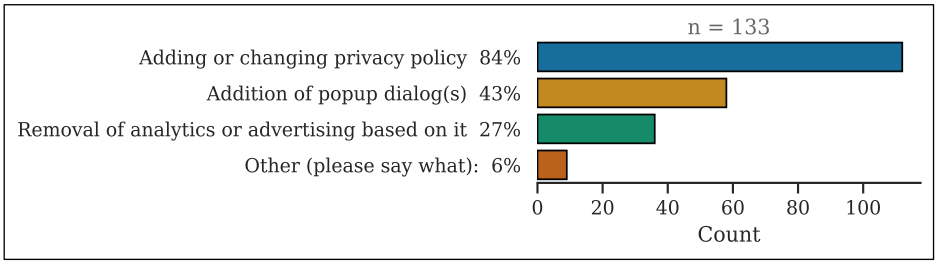

Figure 12: Changes Due to GDPR

Table 1: Pearson R Results (R, P) for Developer Survey Security Scores

Independent: Dependent:	Expertise Support	Requirements	Developer Knowledge	Assurance Technique Use
Assurance Technique Use	0.56, 3.9e-25	0.37, 1.5e-11	0.27, 8.6e-07	
Security Update Frequency	0.16, 0.0085	0.25, 2e-05	*0.03, 0.61*	0.41, 5.7e-13

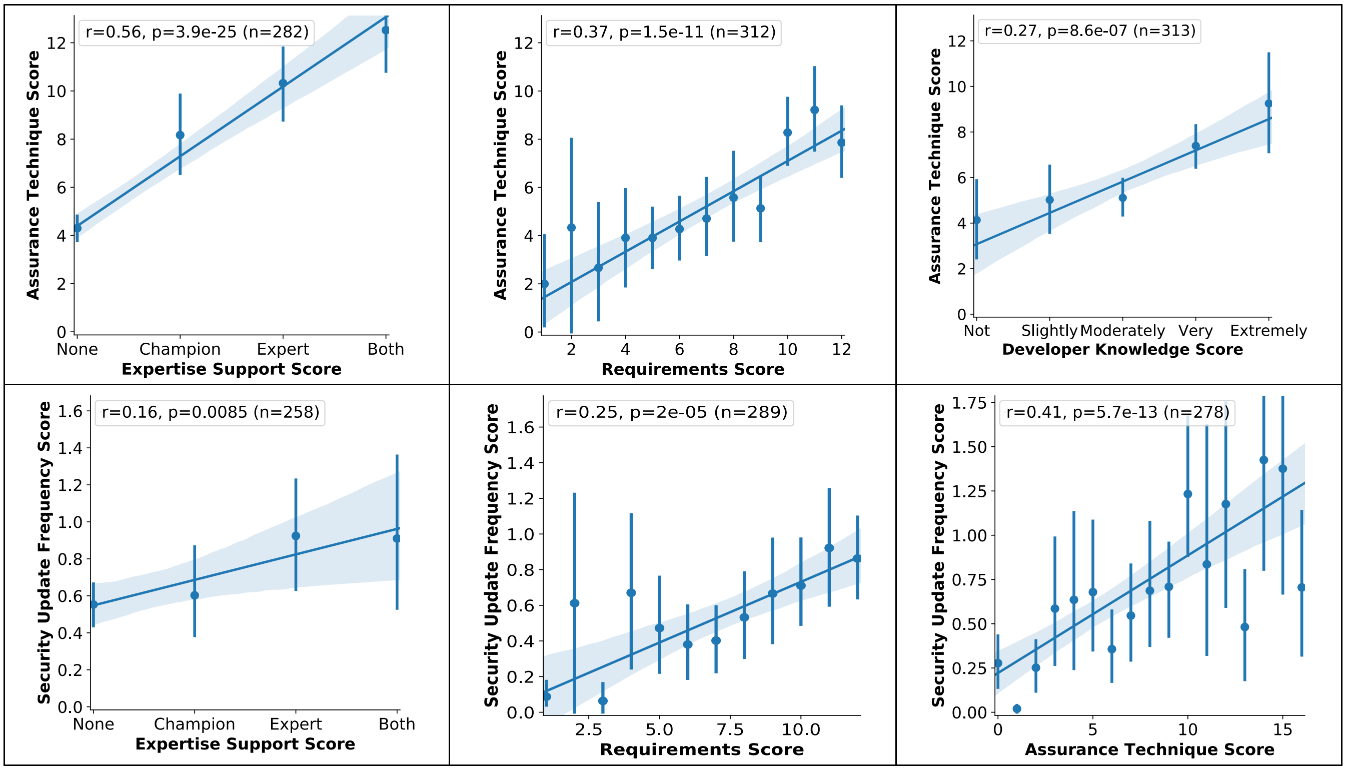

Figure 13: Cross-plots of the Scores with Significant Correlations

Figure 13 shows x-y plots of these significant results. Dots and vertical bars show the mean and its 95% confidence interval for the y-readings corresponding to each x-value. The plots also show a simple linear regression line and its confidence limits. The graphs validate the preconditions for the use of Pearson R [35]: particularly homoscedascity and lack of outliers.

5.5. Post-Hoc Justification for Score Calculation and Analysis

We observe that the first two plots also justify our choice of the calculation for the Requirements Score and Expertise Support Score since the use of assurance techniques shows a strong linear relationship to both scores.

For each of the six pairs of values highlighted in Table 1, we compared Decision Tree models with the corresponding linear models. (F-Test, with a cut-off alpha 0.01). We found no significant differences between the six pairs of models, which justifies using the simpler Pearson R (linear) model. See Appendix D for details.

5.6. Findings on Application Security Indications

In the Phase 2 analysis, of the tools used, CogniCrypt reported no issues for 32% of apps; FlowDroid for 35% and the Bad SSL/MalloDroid combination for 70%. Only 20% of apps analyzed showed no issues from any of the tools.

5.7. Linear Analysis of App Analysis Scores

Table 2 shows the results of the analysis described in Section 4.2. It correlates each of three dependent scores representing "fewer security defects" against the four independent "need and mechanisms for security and privacy" scores. Non-italic figures highlighted in yellow indicate a statistically significant result (p<0.01)

Table 2: Pearson R Results (R, P) Correlating App Security Measurements with Developer-based Factors

Independent: Dependent:	Expertise Support	Requirements	Developer Knowledge	Assurance Technique Use
Cryptographic API Misuse	-0.17, 0.016	*-0.06, 0.37*	*-0.09, 0.17*	*-0.13, 0.047*
Privacy Leak	*-0.09, 0.20*	*-0.01, 0.85*	*0.02, 0.81*	*0.02, 0.81*
SSL Security	*-0.14, 0.049*	*0.01, 0.93*	*-0.02, 0.76*	*-0.08, 0.20*

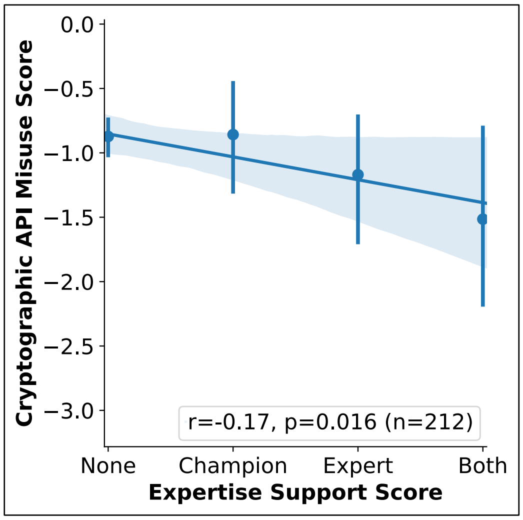

Figure 14: Worse Cryptosecurity with Expert Involvement?

Only one result achieves significance and bizarrely that result suggests a negative correlation: the involvement of security professionals and champions is associated with worse Cryptographic API misuse outcomes.

Figure 14 explores this odd finding. It shows that the effect is not large, and that both experts and champions seem to be associated with the negative correlation, though experts more so. We note, as well, that the p-value is only just significant given the Bonferroni correction (Threshold for significance 0.05/3 = 0.017).

Disappointingly, use of assurance techniques was not associated with better security outcomes, nor was developer security knowledge, nor was a user requirement for good security.

6. Discussion

At first sight, the findings in Sections 5.6 and 5.7 give a depressing view of app security. From Section 5.6 we see that over 80% of apps had reported defects from our analysis tools. From Figure 10 we see that the majority of apps get security updates less than once a year. From the analysis of the app security measurements, Table 2 shows that security outcomes seem to have little correlation with an app's perceived need for security and privacy.

And Figure 12 shows that GDPR's new compliance rules for apps have had little real positive impact. Certainly, in many cases cosmetic changes may have been all that was needed; but the finding suggests that GDPR has not been a strong force to improve app security and privacy.

6.1. Adoption of Security Techniques by Developers

However, there are positive aspects too. Considering the findings in Section 5.2, Figure 7 shows us that the vast majority of the respondents consider themselves to have at least some security knowledge, and thus are likely to be aware of security as a possible issue in their software development. Indeed, Figure 6 shows that more than 60% of the respondents consider security to be very or extremely important to their users, and even more put the same value on privacy.

Section 5.2's combinations of assurance techniques used are particularly interesting in suggesting how security improvement is happening. Though the analysis only covers a small fraction of the total population, those respondents it considers are the ones using only a proportion of the Assurance Techniques and it therefore offers an insight into which techniques are adopted first. One would expect teams whose security is driven by external experts to adopt the Threat Assessment/Penetration Test combination, since both of these activities can be carried out by the experts themselves; actually, rather more adopt tool-only techniques (Auto. Static Analysis and Config. Review), or code-review based techniques (Auto. Static Analysis and Code Review), perhaps because few have access to security experts (Section 5.2).

This suggests that the adoption of assurance techniques is being driven by the developers themselves, rather than by external security experts, and so what we are seeing is developer-led security. This tallies with the reasons given for app security changes in Figure 11, where the most common reason for changes was developer initiative. It also corresponds to the views of security experts, who emphasize the importance of developer initiative in improving software security [53].

6.2. Appropriate Use of Security Techniques

Using security assurance techniques usually has a cost, both in time and in financial terms [45], and therefore it is poor economics to adopt them in cases where they are not required. From Table 1 we see that this is correctly reflected in the Android ecosystem: the use of Assurance Techniques increases in line with the importance of security for the app. We suggest that the correlation with the involvement of security professionals/champions and with developer knowledge of security may be an effect (expert developers and security professionals will tend to work on products that need security) as much as a cause (their involvement causes increased assurance technique use).

Updating apps also has a considerable cost, and again we would anticipate having more security updates in cases where security is important for the app. Again Table 1 confirms this behavior, and shows that, justifiably, there is no correlation between the security update frequency and the security experience of the developer.

6.3. Impact on Real App Security

It was disappointing that the use of assurance techniques did not appear to be a major factor leading to better security outcomes when we analyzed the apps themselves. Even though the analysis tools can only detect a limited range of code level security issues, we expected more security-experienced developers and those using assurance techniques—especially Static Code Analysis—to generate fewer such issues.

We conclude that other factors must drown out this effect. We observe, for example, that most app binary code will consist of libraries, and even up-to-date libraries will differ enormously in the number of such issues they may have. We hypothesize that the scores generated by the tools we used depend more on the nature of the libraries needed to implement the app functionality than on any attributes of the non-library code created by the developers; current tools cannot verify this effect (Section 4.5).

More surprising is the finding that the involvement of professionals and champions seems to be associated with increased numbers of Cryptographic API issues. It seems unlikely that this is because they create the issues. Instead, we observe that our tools will not detect a failure to use cryptography in apps where it is required, whereas experts or champions will do so. We suggest that teams involving experts or champions will therefore tend to use cryptography more frequently, leading to more such issues.

7. Summary and Conclusions

This paper describes the creation and deployment of a survey to Android app developers, in which we asked them a range of questions related to their approach to security and privacy in app development; and a second phase in which we compared the answers with the outcomes of running security analysis tools on one of their apps. The research addresses the questions as follows:

RQ1: *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in the development team?*

From the 335 survey responses analyzed, we found a high level of reported security need for the app development, but less use of practical security assurance techniques (Section 5.2). Where such techniques were used, this was in proportion to the perceived need, as was the involvement of professionals and security champions. The frequency of app security updates followed a similar pattern (Sections 5.4, 6.2).

Considering the "how" of RQ1: in the perception of respondents to the survey, app security improvements have been predominantly driven by developers themselves (Section 6.1); this is supported by the observation that the assurance techniques first adopted are those most easily available to developers. GDPR has also had an impact, though the resulting changes for GDPR have been mainly cosmetic (Section 5.3).

RQ2: *To what extent do the need for security, the involvement of specialist roles, and the use of assurance techniques in a development team lead to fewer security defects?*

The results of the app analysis showed little relationship with the reported security drivers and development process from the survey; we believe this reflects the inability of the current generation of binary analysis tools to analyze libraries effectively and separately from the main app code. We did however find the involvement of security specialists or champions to be associated with more Cryptographic API issues, probably since they correctly enforce much more Cryptography use (Sections 5.7, 6.3)

RQ3 *What proportion of Android developers have access to security experts*?

Section 5.2 concludes that between 14% and 22% of developers work with security experts.

RQ4 *To what extent do Android developers actually use assurance techniques?*

Only between 22% and 30% regularly use assurance techniques (Section 5.2)

Further, contrasting the high need for security with the low use of assurance techniques and low availability of security professionals, we suggest that there is an urgent need for ways to support app developers in adopting security assurance techniques in the absence of security professionals.

7.1. Future Work

As Section 6.3 discusses, we need binary analysis tools capable of:

1. Detecting library versions
2. Performing static analysis on library components separately from the main code.

This is an active area of research; once such tools are available, a further survey using these will provide both valuable results, and an indication of changes over time in Android developer security practices.

More information is also needed to support developers in using these assurance techniques, starting with how developers currently use each one. Specific questions might address where developers go to get security advice; what tools they use to analyze their code; the methods they use for library analysis; how they approach penetration testing; what forms of code review they use; and how they tackle threat assessment. A further online survey can investigate these questions.

7.2. Notes and Credits

A privacy-preserving set of the survey data, along with the full questions and data description, is available online [52]

First, we thank Christian Stransky of LU Hannover for obtaining the Google Play data and APK files used as a basis for the survey; and Dominik Wermke of LU Hannover for initiating the use of Python and Jupyter notebooks for statistical analysis in this project.

We thank Dr Tamara Lopez of the Open University, UK, for her helpful review of the survey questionnaire; Dr Yasemin Acar, of LU Hannover for practical guidance on creating and validating questionnaires; and Professor Ian White, of UCL, UK, for valuable advice on the statistical analysis.

We also thank the eight anonymous reviewers of this and an earlier version of this paper, who have all contributed significantly; and particularly USENIX shepherd Professor Daniel Zappala of Brigham Young University.

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972).

8. References

[1] Acar, Y., Backes, M., Fahl, S., et al. Comparing the Usability of Cryptographic Apis. *2017 IEEE Symposium on Security and Privacy (SP)*, IEEE (2017), 154–171.

[2] Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L., and Stransky, C. You Get Where You're Looking For: The Impact of Information Sources on Code Security. *IEEE Symposium on Security and Privacy*, (2016), 289–305.

[3] Anscombe, F.J. The Transformation of Poisson, Binomial and Negative-Binomial Data. *Biometrika 35*, 3/4 (1948), 246.

[4] Arzt, S., Rasthofer, S., Fritz, C., et al. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, (2014).

[5] Assal, H. and Chiasson, S. Think Secure From the Beginning: A Survey With Software Developers. *Conference on Human Factors in Computing Systems (CHI)*, (2019).

[6] Backes, M., Bugiel, S., and Derr, E. Reliable Third-Party Library Detection in Android and Its Security Applications. *Proceedings of the ACM Conference on Computer and Communications Security*, (2016), 356–367.

[7] Bai, J., Wang, W., Qin, Y., Zhang, S., Wang, J., and Pan, Y. BridgeTaint: A Bi-Directional Dynamic Taint Tracking Method for JavaScript Bridges in Android Hybrid Applications. *IEEE Transactions on Information Forensics and Security 14*, 3 (2019), 677–692.

[8] Becker, I., Parkin, S., and Sasse, M.A. Finding Security Champions in Blends of Organisational Culture. *Proceedings 2nd European Workshop on Usable Security*, (2017).

[9] Bell, L., Brunton-Spall, M., Smith, R., and Bird, J. *Agile Application Security: Enabling Security in a Continuous Delivery Pipeline*. O'Reilly, Sebastopol, CA, 2017.

[10] Caputo, D.D., Pfleeger, S.L., Sasse, M.A., Ammann, P., Offutt, J., and Deng, L. Barriers to Usable Security? Three Organizational Case Studies. *IEEE Security and Privacy 14*, 5 (2016), 22–32.

[11] CONSORT. Checklist of Information to Include When Reporting a Randomized Trial. 2010, 11–12. http://www.consort-statement.org/consort-2010.

[12] Coopamootoo, K.P.L. and Gross, T. *A Codebook for Evidence-Based Research: The Nifty Nine Completeness Indicators*. Newcastle, 2017.

[13] Date, S. The F-Test for Regression Analysis - Towards Data Science. https://towardsdatascience.com/fisher-test-for-regression-analysis-1e1687867259.

[14] Deborah J. Rumsey. *Statistics Essentials For Dummies*. Wiley, For Dummies, 2019.

[15] Derr, E., Bugiel, S., Fahl, S., Acar, Y., and Backes, M. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, ACM Press (2017), 2187–2200.

[16] Egelman, S. and Peer, E. Scaling the Security Wall : Developing a Security Behavior Intentions Scale (SeBIS). *Conference on Human Factors in Computing Systems (CHI2015)*, (2015).

[17] Eichberg, M. and Hermann, B. A Software Product Line for Static Analyses: The OPAL Framework. *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) 2014-June*, June (2014).

[18] Enck, W., Octeau, D., McDaniel, P., and Chaudhuri, S. A Study of Android Application Security. *Proceedings of the 20th USENIX conference on Security*, (2011).

[19] European Commission. General Data Protection Regulation (GDPR). 2019. https://ec.europa.eu/info/law/law-topic/data-protection_en.

[20] Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., and Freisleben, B. Why Eve and Mallory Love Android: An Analysis of Android SSL Security Categories and Subject Descriptors. *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, ACM Press (2012).

[21] Fowler, F.J. *Survey Research Methods*. Sage.

[22] Glanz, L., Amann, S., Eichberg, M., et al. CodeMatch: Obfuscation Won't Conceal Your Repackaged App. *Proceedings of ESEC/FSE'17*, (2017), 638–648.

[23] Haney, J.M. and Lutters, W.G. The Work of Cybersecurity Advocates. *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, ACM Press (2017), 1663–1670.

[24] Kline, T. Classical Test Theory: Assumptions, Equations, Limitations, and Item Analyses. In *Psychological Testing: A Practical Approach to Design and Evaluation*. SAGE Publications, Inc., Thousand Oaks, California, 2005.

[25] Kluyver, T., Ragan-kelley, B., Pérez, F., et al. Jupyter Notebooks: A Publishing Format for Reproducible Computational Workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016, 87–90.

[26] Kruger, S., Nadi, S., Reif, M., et al. CogniCrypt: Supporting Developers in Using Cryptography. *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, (2017), 931–936.

[27] Li, L., Bartel, A., Bissyandé, T.F., et al. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. *Proceedings - International Conference on Software Engineering 1*, (2015), 280–291.

[28] Li, L., Bissyandé, T.F., Papadakis, M., et al. Static Analysis of Android Apps: A Systematic Literature Review. *Information and Software Technology 88*, (2017), 67–95.

[29] McDaniel, P. and Enck, W. Not So Great Expectations: Why Application Markets Haven't Failed Security. *IEEE Security & Privacy Magazine 8*, 5 (2010), 76–78.

[30] Nayak, K., Marino, D., Efstathopoulos, P., and Dumitraş, T. Some Vulnerabilities Are Different Than Others: Studying Vulnerabilities and Attack Surfaces in the Wild. *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, (2014).

[31] O'Brien, R.M. The Use of Pearson's with Ordinal Data. *American Sociological Review 44*, 5 (1979), 851–857.

[32] Oliveira, D., Rosenthal, M., Morin, N., Yeh, K.-C., Cappos, J., and Zhuang, Y. It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC14)*, (2014).

[33] Oltrogge, M., Derr, E., Stransky, C., et al. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. *Proceedings - IEEE Symposium on Security and Privacy*, IEEE (2018), 634–647.

[34] OWASP. Mobile Security Project - Top Ten Mobile Risks. https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks.

[35] Pal, S. The Assumptions in Linear Correlations. *Helpful Stats*, 2017. https://helpfulstats.com/assumptions-correlation/.

[36] Presser, S., Couper, M.P., Lessler, J.T., et al. Methods for Testing and Evaluating Survey Questions. *Public Opinion 68*, 1 (2004), 109–130.

[37] Qualtrics. Qualtrics Survey Service. https://www.qualtrics.com/.

[38] Rasthofer, S., Arzt, S., Hahn, R., Kolhagen, M., and Bodden, E. Black Hat 2015: (In)Security of Backend-as-a-Service. 2015. http://bodden.de/pubs/rah+15backend.pdf.

[39] Reyes, I., Wijesekera, P., Reardon, J., et al. Won't Somebody Think of the Children? Examining COPPA Compliance at Scale. *Proceedings on Privacy Enhancing Technologies 2018*, 3 (2018), 63–83.

[40] Rumsey, D. *Statistics II for Dummies*. Wiley, Indianapolis, 2009.

[41] Safavian, S.R. and Landgrebe, D. A Survey of Decision Tree Classifier Methodology. *IEEE Transactions on Systems, Man and Cybernetics 21*, 3 (1991), 660–674.

[42] Senarath, A. and Arachchilage, N.A.G. Why Developers Cannot Embed Privacy into Software Systems? *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE18)*, (2018), 211–216.

[43] Stack Overflow. Developer Survey Results 2019. 2019. https://insights.stackoverflow.com/survey/2019.

[44] Stevens, S.S. On the Theory of Scales of Measurement. *Science 103*, 2684 (1946), 677–680.

[45] Such, J.M., Gouglidis, A., Knowles, W., Misra, G., and Rashid, A. Information Assurance Techniques: Perceived Cost Effectiveness. *Computers and Security 60*, (2016), 117–133.

[46] The Harris Poll. *Norton LifeLock Cyber Safety Insights Report*. 2018.

[47] Turpe, S. The Trouble with Security Requirements. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, (2017), 122–133.

[48] USCF. Confidence Interval for a Proportion. http://www.sample-size.net/confidence-interval-proportion/.

[49] Vaniea, K. and Rashidi, Y. Tales of Software Updates: The Process of Updating Software. *Proceedings for Computer Human Interaction (CHI) 2016*, (2016), 3215–3226.

[50] Wei, F., Lin, X., Ou, X., Chen, T., and Zhang, X. JN-SAF: Precise and Efficient NDK/JNI-Aware Inter-Language Static Analysis Framework for Security Vetting of Android Applications With Native Code. *Proceedings of the ACM Conference on Computer and Communications Security (CCS18)*, 1 (2018), 1137–1150.

[51] Weir, C., Becker, I., Noble, J., Blair, L., Sasse, M.A., and Rashid, A. Interventions for Software Security: Creating a Lightweight Program of Assurance Techniques for Developers. *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, IEEE (2019).

[52] Weir, C., Hermann, B., Stransky, C., Wermke, D., and Fahl, S. Public Dataset from Online Android App Developer Survey. 2019. https://dx.doi.org/10.17635/lancaster/researchdata/319.

[53] Weir, C., Rashid, A., and Noble, J. I'd Like to Have an Argument, Please: Using Dialectic for Effective App Security. *Proceedings 2nd European Workshop on Usable Security*, Internet Society (2017).

[54] Wermke, D., Reaves, B., Huaman, N., Traynor, P., Acar, Y., and Fahl, S. A Large Scale Investigation of Obfuscation Use in Google Play. *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, (2018), 222–235.

[55] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., and Joosen, W. On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared. *Information and Software Technology 51*, 7 (2009), 1152–1171.

[56] Witschey, J., Zielinska, O., Welk, A., Murphy-Hill, E., Mayhorn, C., and Zimmermann, T. Quantifying Developers' Adoption of Security Tools. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, ACM Press (2015), 260–271.

[57] Xie, J., Lipford, H.R., and Chu, B.B.-T. Evaluating Interactive Support for Secure Programming. *SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 2707–2716.

[58] Zhao, Q., Zuo, C., Pellegrino, G., and Lin, Z. Geo-locating Drivers : A Study of Sensitive Data Leakage in Ride-Hailing Services. *Symposium Network and Distributed System Security Symposium (NDSS)*, February (2019).

Appendix A Analysis Tool Versions

The following are the versions of the tools we used for application analysis.

MalloDroid	Version Dec 30, 2013
OPAL framework	Version 1.0.0
curl	Version 7.64.0
openssl	Version 1.1.1b
FlowDroid	Version 2.7.1
LibScout	Version 2.3.2
CogniCrypt	Version 1.0.0

Appendix B Survey Questions

The following are the survey questions. Some questions were skipped if appropriate (marked with *). The answer formats are abbreviated as follows:

YN	Yes or No
SS	Single Selection.
MS	Multiple Selection
LSS	Likert-Style Scale: Extremely, though to Not at all.
0-100	Slider selecting an integer
N	Integer

In addition, '?' indicates an 'I don't know' option, and 'O' an 'Other' option, where the participant could enter open text. In Q10 and Q21, the option descriptions give the encodings used in Appendix C .

Q1-Q3 were text-only statements.

Q4 Are you working in a team with others, such as developers, testers, project managers? [YN]

Q5* What is your role? [SSO?]
Programmer, Tester, Project Manager, Non-Specific

Q6* What other roles apart from yourself are there in your team? [MS?]
Programmer, Tester, Project Manager, Non-Specific

Q7* About how many people (including developers, project managers, testers) are there in your team? [N]

Q8 Please select all the ways you use to develop Android apps [MSO]
Native Java, JavaScript, C#, Dart, Python, Kotlin, Lua, Native C++

Q10 How often did you release a new version of your app over the past two years? Please give your best estimate; if you have more than one app, please answer for that app that was most frequently updated. [SS]
Never (0), Annually (1), Quarterly (4), Monthly (12), More frequently (24)

Q11* Over the last one to two years, what content has been in your app updates (%)?
New features [0-100]
Non-security bug fixes [0-100]
Security bug fixes [0-100]
Third party library updates [0-100]
Regular maintenance and refactoring [0-100]

Q12 How important is each of the following for your app(s)?
Runs on many different devices [LSS]
Secure against malicious attackers [LSS]
Protects users' privacy [LSS]
Easy to use [LSS]
Supports many features [LSS]
Runs smoothly [LSS]

Q13 How important is security for sales? [LSS]

Q14 How knowledgeable do you consider yourself about information security? [LSS]

Q15 Does your app development ever get support from professional security experts? [YN?]

Q16* Who are these professional security experts (on team/external)? [SS]

Q17* What support do you get from them? Please select all that apply [MSO]

Penetration testing
Audits
Working on team
Security training
Design reviews
I don't know

Q18* About how often do you get support from them? [SS?]
Continuously, Weekly, Monthly, Quarterly, Yearly

Q19 Which of the following have led to changes in the security of your app(s) in the past one to two years? [MSO]

Decision from management
Security crisis within your organization
Media coverage about app security
Something bad happening to a competitor
Pressure from a partner company
Drive from product or sales team
Pressure from customers
Developer initiative
GDPR requirements
Something bad almost happening to your organization

Q20* What changes have you made as a result of GDPR requirements? [MSO]

Addition of popup dialog(s)
Removal of analytics or advertising based on it
Adding or changing privacy policy

Q21 How much do you use each of the following techniques to find security problems? [SS for each:
Every build (4), Every release (3), Once or occasionally (2), Decided not to use (1), Haven't considered it (0).]

Producing a threat assessment for the app
Scanning code with an automatic code review tool
Using a tool to scan for libraries with known vulnerabilities
Code review by someone other than the developer
Penetration testing

Q22 What other techniques do you use (if any)? [O]

23 Do you have a security champion within your team? A security champion -- or security hobbyist -- is a non-expert, who takes a particular interest in security. [YN?]

Q24 For how many years have you been developing Android apps? [N]

Q25 For how many years have you been programming in general (not just for Android)? [N]

Q26 About how many Android apps have you helped develop in total? [N]

Q27 Is developing Android apps your primary job? [YN]

Q28 Have you contributed to an open source project in the past year? [YN]

Q29 To which gender identity do you most identify? [SS]:
Female, Non-binary, Male, Prefer not to say

Q30 What is the main spoken language you use at work? [SS]
English, Chinese, Spanish, Arabic, German, French, Other

Q31 In which country do you currently reside? [SS]

Appendix C Calculation of Scores

This section describes how scores were calculated from the survey answers.

Likert-Style Scales were encoded as:
Extremely … (4), Very … (3), Moderately … (2), Slightly …(1), Not … at all (0)

Assurance Technique Score: sum of all five sub-questions of Q21, each encoded as shown.

Developer Knowledge Score: LSS encoding of Q14

Expertise Support Score: as the following table.

Q23: \ Q15:	No	Yes
No	0	2
Yes	1	3

Requirements Score: sum of LSS encodings for Q12 (Secure against malicious attackers), Q12 (Protects users' privacy) and Q13

Security Update Frequency Score: This required an Update Frequency Estimate of Q10 encoded as shown multiplied by Q11 (Security bug fixes) and divided by 100. The score was Log (this value plus 1).

Appendix D Model Comparison

To compare a decision tree model, we used the Python scikit-learn library's DecisionTreeRegressor, compared with StatsModels' OLS (Ordinary Least Squares).

We compared each pair of models using the F-Test calculation [13], taking the number of 'leaf nodes' in the decision tree as the degrees of freedom for that model in the F-Test. Applying the Bonferroni correction [40], we took the required Alpha P-value for significance as 0.01. The calculated P-values values ranged from 0.2 to 0.5, and did not approach that value.