Security in the Software Development Lifecycle

Hala Assal and Sonia Chiasson, ***Carleton University***

https://www.usenix.org/conference/soups2018/presentation/assal

This paper is included in the Proceedings of the Fourteenth Symposium on Usable Privacy and Security.

August 12–14, 2018 • Baltimore, MD, USA

ISBN 978-1-939133-10-6

Open access to the Proceedings of the Fourteenth Symposium on Usable Privacy and Security is sponsored by USENIX.

Security in the Software Development Lifecycle

Hala Assal
School of Computer Science
Carleton University
Ottawa, ON Canada
HalaAssal@scs.carleton.ca

Sonia Chiasson
School of Computer Science
Carleton University
Ottawa, ON Canada
Chiasson@scs.carleton.ca

ABSTRACT

We interviewed developers currently employed in industry to explore real-life software security practices during each stage of the development lifecycle. This paper explores steps taken by teams to ensure the security of their applications, how developers' security knowledge influences the process, and how security fits in (and sometimes conflicts with) the development workflow. We found a wide range of approaches to software security, if it was addressed at all. Furthermore, real-life security practices vary considerably from best practices identified in the literature. Best practices often ignore factors affecting teams' operational strategies. *Division of labour* is one example, whereby complying with best practices would require some teams to restructure and re-assign tasks—an effort typically viewed as unreasonable. Other influential factors include company culture, security knowledge, external pressure, and experiencing a security incident.

1. INTRODUCTION

Software security focuses on the resistance of applications to malicious attacks resulting from the exploitation of vulnerabilities. This is different from security functions, which can be expressed as functional requirements, such as authentication [60]. With increasing connectivity and progress towards the Internet of Things (IoT), threats have changed [30]. In addition to vulnerabilities in traditional computing systems (*e.g.*, Heartbleed [21]), vulnerabilities are found in devices and applications that are not necessarily considered security sensitive, such as cars [28], and medical devices [43]. Moreover, the threat is no longer limited to large enterprises; Small and Medium Enterprises (SMEs) are increasingly becoming targets of cyberattacks [50].

With increasing threats, addressing security in the Software Development Lifecycle (SDLC) is critical [25, 54]. Despite initiatives for implementing a *secure* SDLC and available literature proposing tools and methodologies to assist in the process of detecting and eliminating vulnerabilities (*e.g.* [16, 18, 20, 48]), vulnerabilities persist. Developers are often viewed as "the weakest link in the chain" and are blamed for security vulnerabilities [27, 58]. However, simply expecting developers to keep investing more efforts in security is unrealistic and unlikely to be fruitful [14].

Usable security research focusing on developers and the human factors of software security–a new area that has not been sufficiently investigated–has the potential for a widespread positive influence on security [14, 27]. Towards guiding research in this area, Acar *et al.* [14] proposed a research agenda for usable security for developers where they highlight important research questions.

Our work is a step towards addressing one of the prominent research areas outlined by Acar *et al.*'s research agenda [14]. This paper explores steps that teams are taking to ensure the security of their applications, how developers' security knowledge influences the process, and how security fits in (and sometimes conflicts with) the development workflow. We interviewed 13 developers who described their tasks, their priorities, as well as tools they use. During the data analysis we recognized that our participants' practices and attitudes towards security formed two groups, each with trends distinguishable from the other group. On comparing real-life security practices to best practices, we also found significant deviations.

This paper makes the following contributions.

- We present a qualitative study looking at real-life practices employed towards software security.
- We amalgamate software security best practices extracted from the literature into a concise list to assist further research in this area.
- We reflect on how well current security practices follow best practices, identify significant pitfalls, and explore why these occur.
- Finally, we discuss opportunities for future research.

2. RELATED WORK

Green and Smith [27] discussed how research addressing the human factors of software security is generally lacking, and that developers are often viewed as "the weakest link"—mirroring the early attitude towards end-users before usable security research gained prominence. While developers are more technically experienced than typical end-users, they should not be mistaken for security experts [14, 27]. They need support when dealing with security tasks, *e.g.*, through developer-friendly security tools [58] or programming languages that prevent security errors [27]. To this end, Acar *et al.* [14] outlined a research agenda towards understanding developers' attitudes and security knowledge,

USENIX Symposium on Usable Privacy and Security (SOUPS) 2018.
August 12–14, 2018, Baltimore, MD, USA.

exploring the usability of available security development tools, and proposing tools and methodologies to support developers in building secure applications. We now discuss relevant research addressing such human aspects of software security.

Generally, studies in this area face challenges in recruiting developers and ensuring ecologically validity. Developers are busy and must often comply with organizational restrictions on what can be shared publicly. To partially address these issues, Stransky *et al.* [51] designed a platform to facilitate distributed online programming studies with developers.

Oliveira *et al.* [22] showed that security vulnerabilities are "blind spots" in developers' decision-making processes; developers mainly focus on functionality and performance. To improve code security, Wurster and van Oorschot [58] recommend taking developers out of the development loop through the use of Application Programming Interfaces (APIs). Towards this goal, Acar *et al.* [12] evaluated five cryptographic APIs and found usability issues that sometimes led to insecure code. However, they found that documentation that provided working examples was significantly better at guiding developers to write secure code. Focusing on software security resources in general, Acar *et al.* [15] found that some available security advice is outdated and most resources lack concrete examples. In addition, they identified some underrepresented topics, including program analysis tools.

Focusing on security analysis, Smith *et al.* [48] showed that tools should better support developers' information needs. On exploring developers' interpretation of Static-code Analysis Tool (SAT) warnings, they found that participants frequently sought additional information about the software ecosystem and resources. To help developers to focus on the overall security of their code, Assal *et al.* [16] proposed a visual analysis environment that supports collaboration while maintaining the codebase hierarchy. This allows developers to build on their existing knowledge of the codebase during code analysis. Perl *et al.* [41] used machine learning techniques to develop a code analysis tool. Their tool has significantly fewer false-positives compared to similar ones. Nguyen *et al.* [40] developed a plugin to help Android application developers adhere to, and learn about, security best practices without distributing their workflow.

Despite their benefits [17], SATs are generally underused [31]. Witschey *et al.* [56] investigated factors influencing the adoption of security tools, such as tool qualities, and developers' personalities and experiences. They found that more experienced developers are more likely to adopt security tools, whereas tool complexity was a deterring factor. Additionally, Xiao *et al.* [59] found that the company culture, the application's domain, and the company's standards and policies were among the main determinants for the developers' adoption of security tools. To encourage developers to use security tools, Wurster and van Oorschot [58] suggest mandating their use and rewarding developers who code securely.

As evidenced, several research gaps remain in addressing the human aspects of software security. Our study takes a holistic perspective to explore real-life security practices, an important step in improving the status-quo.

3. STUDY DESIGN AND METHODOLOGY

We designed a semi-structured interview study and received IRB clearance. The interviews targeted 5 main topics: general development activities, attitude towards security, security knowledge, security processes, and software testing activities (see Appendix A for interview script). To recruit participants, we posted on development forums and relevant social media groups, and announced the study to professional acquaintances. We recruited 13 participants; each received a $20 Amazon gift card for participation. Before the one-on-one interview, participants filled out a demographics questionnaire. Each interview lasted approximately 1 hour, was audio recorded, and later transcribed for analysis. Interviews were conducted in person ($n = 3$) or through VOIP/video-conferencing ($n = 10$). Data collection was done in 3 waves, each followed by preliminary analysis and preliminary conclusions [26]. We followed Glaser and Strauss's [26] recommendation by concluding recruitment on saturation (*i.e.*, when new data collection does not add new themes or insights to the analysis).

Teams and participants. A project team consist of teams of developers, testers, and others involved in the SDLC. Smaller companies may have only one project team, while bigger companies may have different project teams for different projects. We refer to participants with respect to their project teams; team i is referred to as Ti and P-Ti is the participant from this team. We did not have multiple volunteers from the same company. Our data contains information from 15 teams in 15 different companies all based in North America; one participant discussed work in his current (T7) and previous (T8) teams, another discussed his current work in T10 and his previous work in T11. In our dataset, seven project teams build web applications and services, such as e-finance, online productivity, online booking, website content management, and social networking. Eight teams deliver other types of software,*e.g.*, embedded software, kernels, design and engineering software, support utilities, and information management and support systems. This classification is based on participants' self-identified role and products with which they are involved, and using Forward and Lethbridge's [24] software taxonomy. Categorizing the companies to which our teams belong by number of employees [19], 7 teams belong to SMEs (T4, T7, T10–T14) and 8 teams belong to large enterprises (T1–T3, T5, T6, T8, T9, T15). All participants hold university degrees which included courses in software programming, and are currently employed in development with an average of 9.35 years experience ($Md = 8$). We did not recruit for specific software development methodologies. Some participants indicated following a waterfall model or variations of Agile. See Table 3 in Appendix B for participant demographics.

Analysis. Data was analyzed using the Qualitative Content Analysis methodology [9,23]. It can be deductive, inductive, or a combination thereof. For the deductive approach, the researcher uses her knowledge of the subject to build an analysis matrix and codes data using this matrix [9]. The inductive method, used when there is no existing knowledge of the topic, includes open coding, identifying categories, and abstraction [9].

We employed both the deductive and inductive methods of content analysis. The deductive method was used to structure our analysis according to the different development stages. We built an initial analysis matrix of the main SDLC stages [49]. After a preliminary stage of categoriz-

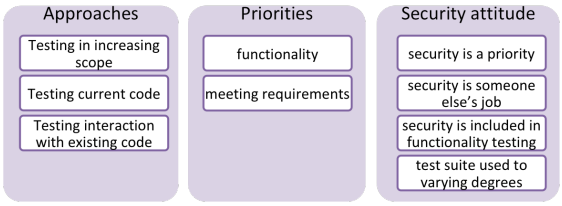

Figure 1: Security adopters: developer testing abstraction

ing interview data and discussions between the researchers, the matrix was refined. The final analysis matrix defines the stages of development as follows. *Design* is the stage where the implementation is conceptualized and design decisions are taken; *Implementation* is where coding takes place; *Developer testing* is where testing is performed by the developer; *Code analysis* is where code is analyzed using automated tools, such as SATs; *Code review* is where code is examined by an entity other than the developer; *Post-development testing* is where testing and analysis processes taking place after the developer has committed their code.

We coded interview data with their corresponding category from the final analysis matrix, resulting in 264 unique excerpts. Participants talked about specific tasks that we could map to the matrix stages, despite the variance in development methodologies. We then followed an inductive analysis method to explore practices and behaviours within each category (development stage) as recommended by the content analysis methodology. We performed open coding of the excerpts where we looked for interesting themes and common patterns in the data. This resulted in 96 codes. Next, data and concepts that belonged together were grouped, forming sub-categories. Further abstraction of the data was performed by grouping sub-categories into generic categories, and those into main categories. The abstraction process was repeated for each stage of development. As mentioned earlier, during our analysis we found distinct differences in attitudes and behaviours that were easily distinguishable into two groups, we call them *the security adopters* and *the security inattentive*. We thus present the emerging themes and our analysis of the two groups independently. Figure 1 shows an example of the abstraction process for developer testing data for the security adopters. While all coding was done by a single researcher, two researchers met regularly to thoroughly and collaboratively review and edit codes, and group and interpret the data. To verify the reliability of our coding, we followed best practices by inviting a researcher who has not been involved with the project to act as a second coder, individually coding 30% of the data. We calculated Krippendorff's alpha [33] to assess inter-rater reliability, and $\alpha = 0.89$ (percentage of agreement = 91%). According to Krippendorff [34], *alpha* ≥ 0.80 indicates that coding is highly reliable and that data is "similarly interpretable by researchers". In case of disagreements, we had a discussion and came to an agreement on the codes.

Limitations: Our study included a relatively small sample size, thus generalizations cannot be made. However, our sample size followed the concept of saturation [26]; participant recruitment continued until no new themes were emerging. Additionally, recruiting participants through personal contacts could result in biasing the results. While we cannot guarantee representativeness of a larger population, the interviewer previously knew only 3/13 participants. The remaining ten participants were previously unknown to the researcher and each represented a different company. While interviews allowed us to explore topics in depth, they presented one perspective on the team. Our data may thus be influenced by participants' personal attitudes and perspectives, and may not necessarily reflect the whole team's opinions. However, we found that participants mainly described practices as encouraged by their teams.

Table 1: The degree of security in the SDLC. ● : secure, ○ : somewhat secure, × : not secure, ⊗ : not performed, ? : no data

(a) The Security Adopters

	Design	Implementation	Developer testing	Code analysis	Code review	Post-dev testing
T1	×	●	×	●	●	●
T3	?	●	?	●	●	●
T5	●	●	○	●	●	●
T11	?	●	○	●	●	?
T12	×	●	○	●	●	●
T14	×	●	●	⊗	●	●

(b) The Security Inattentive

	Design	Implementation	Developer testing	Code analysis	Code review	Post-dev testing
T2	×	●	×	○	○	●
T4	○	○	○	⊗	○	○
T6	×	○	×	×	○	○
T7	×	×	×	⊗	×	○
T8	×	×	×	⊗	×	●
T9	○	●	○	⊗	○	○
T10	○	○	×	⊗	○	⊗
T13	×	●	×	⊗	○	●
T15	×	×	×	×	×	×

4. RESULTS: SECURITY IN PRACTICE

We assess the degree of security integration in each stage of the SDLC as defined by our final analysis matrix. As mentioned earlier, we found differences in participants' attitudes and behaviours towards security that naturally fell into two distinct groups. We call the first group the *security adopters*: those who consider security in the majority of development stages (at least four stages out of six[1]). The second group who barely considered security or did not consider it at all form the *security inattentive*. We chose the term *inattentive*, as it encompasses different scenarios that led up to poor security approaches. These could be that security was considered and dismissed or it was not considered at all, whether deliberately or erroneously. Table 1 presents two heat maps, one for each group identified in our dataset (see Appendix C for more information). We classified practices during a development stage as:

(●) *secure*: when security is actively considered, *e.g.*, when developers avoid using deprecated functions during the implementation stage.
(○) *somewhat secure*: when security is not consistently considered, *e.g.*, when threat analysis is performed only if someone raises the subject.
(×) *not secure*: when security is not considered at all, *e.g.*, when developers do not perform security testing.
(⊗) *not performed*: when a stage is not part of their SDLC (*i.e.*, considered not secure).
(?): when a participant did not discuss a stage during their interview, therefore denoting missing data.

The heat maps highlight the distinction in terms of security between practices described by participants from the security adopters and the security inattentive groups. The overwhelming red and orange heat map for the security inattentive group visually demonstrates their minimal secu-

[1]At least three stages in cases where we have information about four stages only. Note that this is just a numeric representation and the split actually emerged from the data.

rity integration in the SDLC. Particularly, comparing each stage across all teams shows that even though the security adopters are not consistently secure throughout the SDLC, they are generally more attentive to security than the other group. The worst stage for the security inattentive group is *Code analysis*, which is either not performed or lacks security, followed by the *developer testing* stage, where security consideration is virtually non-existent.

We initially suspected that the degree of security integration in the SDLC would be directly proportional to the company size. However, our data suggests that it is not necessarily an influential factor. In fact, T14, the team from the smallest company in our dataset, is performing much better than T6, the team from the largest company in the security inattentive group. Additionally, we did not find evidence that development methodology influenced security practices.

Although our dataset does not allow us to make conclusive inferences, it shows an alarming trend of low security adoption in many of our project teams. We now discuss data analysis results organized by the six SDLC stages defined in our analysis matrix. All participants discussed their teams' security policies, as experienced from their perspectives, and not their personal preferences. Results, therefore, represent the reported perspectives of the developers in each team.

4.1 Exploring practices by development stage

We found that the prioritization of security falls along a spectrum: at one end, security is a main priority, or it is completely ignored at the other extreme. For each SDLC stage, we discuss how security was prioritized, present common trends, and highlight key messages from the interviews. Next to each theme we indicate which group contributed to its emergence: `(SA)` for the security adopters, `(SI)` for the security inattentive, and `(SA/SI)` for both groups. Table 2 provides a summary of the themes.

4.1.1 Design stage

We found a large gap in security practices described by our participants in the design stage. This stage saw teams at all points on the security prioritization spectrum, however, most participants indicated that their teams did not view security as part of this stage. Our inductive analysis revealed three emerging themes reflecting security prioritization, with one theme common to both the security adopters and the security inattentive, and one exclusive to each group.

Security is not considered in the design stage. `(SA/SI)` Most participants indicated that their teams did not apply security best practices in the design stage. Although they did not give reasons, we can infer from our data (as discussed in other stages) that this may be because developers mainly focus on their functional design task and often miss security [22], or because they lack the expertise to address security. As an example of the disregard for security, practices described by one participant from the security inattentive group violates the recommendation of simple design; they intentionally introduce complexity to avoid rewriting existing code, and misuse frameworks to fit their existing codebase without worrying about introducing vulnerabilities. P-T10 explained how this behaviour resulted in a highly complex code, "*Everything is so convoluted and it's like going down rabbit holes, you see their code and you are like 'why did you write it this way?' [...] It's too much different custom code that only those guys understand.*" Such complexity increases the potential for vulnerabilities and complicates subsequent stages [47]; efforts towards evaluating code security may be hindered by poor readability and complex design choices.

Security consideration in the design stage is adhoc. `(SI)` Two developers said their teams identify security considerations within the design process. In both cases, the design is done by developers who are not necessarily formally trained in security. Security issue identification is adhoc, *e.g.*, if a developer identifies a component handling sensitive information, this triggers some form of threat modelling. In T10, this takes the form of discussion in a team meeting to consider worst case scenarios and strategies for dealing with them. In T4, the team self-organizes with the developers with most security competence taking the responsibility for designing sensitive components. P-T4 said, "*Some developers are assigned the tasks that deal with authorization and authentication, for the specific purpose that they'll do the security testing properly and they have the background to do it.*" In these two teams, security consideration in the design stage lies in the hands of the developer with security expertise; this implies that the process is not very robust. If this developer fails to identify the feature as security-sensitive, security might not be considered at all in this stage.

Security design is very important. `(SA)` Contrary to all others, one team formally considers security in this stage with a good degree of care. P-T5 indicated that his team considers the design stage as their first line of defense. Developers from his team follow software security best practices [1, 8, 47], *e.g.*, they perform formal threat modelling to generate security requirements, focus on relevant threats, and inform subsequent SDLC stages. P-T5 explains the advantages of considering security from this early stage, "*When we go to do a further security analysis, we have a lot more context in terms of what we're thinking, and people aren't running around sort of defending threats that aren't there.*"

4.1.2 Implementation stage

Most participants showed general awareness of security during this stage. However, many stated that they are not responsible for security and they are not required to secure their applications. In fact, some developers reported that their companies do not expect them to have any software security knowledge. Our inductive analysis revealed three themes regarding security prioritization in this stage.

Security is a priority during implementation. `(SA/SI)` All security adopters and two participants from the security inattentive group discussed the importance of security during the implementation stage. They discussed how the general company culture encourages following secure implementation best practices and using reliable tools. Security is considered a developer's responsibility during implementation, and participants explained they are conscious about vulnerabilities introduced by errors when writing code.

Developers' awareness of security is expected when implementing. `(SA/SI)` For those prioritizing security, the majority of security adopters and one participant from the security inattentive group are expected to stay up-to-date

on vulnerabilities, especially those reported in libraries or third-party code they use. The manner of information dissemination differs and corroborates previous research findings [59]. Some have a structured approach, such as that described by P-T1, "*We have a whole system. Whenever security vulnerability information comes from a third-party, [a special team follows] this process: they create an incident, so that whoever is using the third-party code gets alerted that, 'okay, your code has security vulnerability', and immediately you need to address it.*" Others rely on general discussions between developers, *e.g.*, when they read about a new vulnerability. Participants did not elaborate on if and how they assess the credibility and reliability of information sources. The source of information could have a considerable effect on security; previous research found that relying on informal programming forums might lead to insecure code [13]. In Xiao *et al.*'s [59] study, developers reported taking the information source's thoroughness and reputation into consideration to ensure trustworthiness.

Security is not a priority during implementation. `(SI)` On the other end of the security prioritization spectrum, developers from the security inattentive group prioritize functionality and coding standards over security. Their primary goal is to satisfy business requirements of building new applications or integrating new features into existing ones. Some developers also follow standards for code readability and efficiency. However, security is not typically considered a developer's responsibility, to the extent that there are no consequences if a developer introduces a security vulnerability in their code. P-T7 explained, "*If I write a bad code that, let's say, introduced SQL injection, I can just [say] 'well I didn't know that this one introduces SQL injection' or 'I don't even know what SQL injection is'. [...] I didn't have to actually know about this stuff [and] nobody told me that I need to focus on this stuff.*" This statement is particularly troubling given that P-T7 has security background, but feels powerless in changing the perceived state of affairs in his team.

Our analysis also revealed that some developers in the security inattentive group have incomplete mental models of security. This led to the following problematic manifestations, which could explain their poor security practices.

Developers take security for granted. `(SI)` We found, aligning with previous research [22], that developers fully trust existing frameworks with their applications' security and thus take security for granted. Our study revealed that these teams do not consider security when adopting frameworks, and it is unclear if, and how, these frameworks' security is ever tested. To partially address this issue, T4 built their own frameworks to handle common security features to relieve developers of the burden of security. This approach may improve security, however verifying frameworks' security is an important, yet missing, preliminary step.

Developers misuse frameworks. `(SI)` Despite their extreme reliance on frameworks for security, developers in T10 do not always follow their recommended practices. For example, although P-T10 tries to follow them, other developers in his team do not; they occasionally overlook or work-around framework features. P-T10 explains, "*I have expressed to [the team] why I am doing things the way I am, because it's correct, it's the right way to do it with this framework. They chose to do things a completely different way, it's completely messed up the framework and their code. They don't care, they just want something that they feel is right and you know whatever.*" Such framework misuse may result in messy code and could lead to potential vulnerabilities [47]. Although frameworks have shown security benefits [52], it is evident that the manner by which some teams are currently using and relying on them is problematic.

Developers lack security knowledge. `(SI)` Developers from the security inattentive group vary greatly in their security knowledge. Some have haphazard knowledge; they only know what they happen to hear or read about in the news. Others have formed their knowledge entirely from practical experience; they only know what they happen to come across in their work. Developers' lack of software security knowledge could explain why some teams are reluctant to rely on developers for secure implementation. P-T7 said, "*I think they kind of assume that if you're a developer, you're not necessarily responsible for the security of the system, and you [do] not necessarily have to have the knowledge to deal with it.*" On the other hand, some developers have security background, but do not apply their knowledge in practice, as it is neither considered their responsibility nor a priority. P-T7 said, "*I recently took an online course on web application security to refresh my knowledge on what were the common attack on web applications [...] So, I gained that theoretical aspect of it recently and play[ed] around with a bunch of tools, but in practice I didn't actually use those tools to test my software to see if I can find any vulnerability in my own code because it's not that much of a priority.*"

Developers perceive their security knowledge inaccurately. `(SI)` We identified a mismatch between developers' perception of their security knowledge and their actual knowledge. Some developers do not recognize their secure practices as such. When asked about secure coding methods, P-T6 said, "*[The] one where we stop [cross-site scripting]. That's the only one I remember I explicitly used. Maybe I used a couple of other things without knowing they were security stuff.*" In some instances, our participants said they are not addressing security in any way. However, after probing and asking more specific questions, we identified security practices they perform which they did not relate to security.

Furthermore, we found that some developers' mental model of security revolves mainly around security functions, such as using the proper client-server communication protocol. However, conforming with previous research [59], it does not include software security. For example, P-T9 assumes that following requirements generated from the design stage guarantees security, saying "*if you follow the requirements, the code is secure. They take those into consideration.*" However, he mentioned that requirements do not always include security. In this case, and especially by describing requirements as a definite security guarantee, the developer may be referring to security functions (*e.g.*, using passwords for authentication) that he would implement as identified by the requirements. However, the developer did not discuss vulnerabilities due to implementation mistakes that are not necessarily preventable by security requirements.

Our study also revealed the following incident which illustrates how **Vulnerability discovery can motivate secu-**

rity (SI) and improve mental models. Developers in T13 became more security conscious after discovering a vulnerability in their application. P-T13 said, "*We started making sure all of our URLs couldn't be manipulated. [..] If you change the URL and the information you are looking at, [at the] server side, we'd verify that the information belongs to the site or the account you are logged in for.*" Discovering this vulnerability was eye-opening to the team; our participant said that they started thinking about their code from a perspective they had not been considering and they became aware that their code can have undesirable security consequences. In addition, this first-hand experience led them to the knowledge of how to avoid and prevent similar threats.

4.1.3 Developer testing stage

Across the vast majority of our participants, whether adopters or inattentive, security is lacking in the developer testing stage. Functionality is developers' main objective; they are blamed if they do not properly fulfil functional requirements, but their companies do not hold them accountable if a security vulnerability is discovered. P-T7 said, "*I can get away with [introducing security bugs] but with other things like just your day-to-day developer tasks where you develop a feature and you introduce a bug, that kind of falls under your responsibility. Security doesn't.*" Thus, any security-related efforts by developers are viewed as doing something extraordinary. For example, P-T2 explained, "*If I want to be the hero of the day [and] I know there's a slight possible chance that these can be security vulnerabilities, [then] I write a test and submit it to the test team.*" We grouped participants' approaches to security during this stage into four categories.

Developers do not test for security. (SA/SI) The priority at this stage is almost exclusively functionality; it increases in scope until the developer is satisfied that their code is fulfilling functional requirements and does not break any existing code. And even then, these tests vary in quality. Some developers perform adhoc testing or simply test as a sanity check where they only verify positive test cases with valid input. Others erroneously, and at times deliberately, test only ideal-case scenarios and fail to recognize worst-case scenarios. The majority of developers do not view security as their responsibility in this stage; instead they are relying on the later SDLC stages. P-T2 said, "*I usually don't as a developer go to the extreme of testing vulnerability in my feature, that's someone else's to do. Honestly, I have to say, I don't do security testing. I do functional testing.*" The participant acknowledged the importance of security testing, however, this task was considered the testing team's responsibility as they have more knowledge in this area.

Security is a priority during developer testing. (SA) As an exception, our analysis of P-T14's interview indicates that his company culture emphasizes the importance of addressing security in this stage. His team uses both automated and manual tests to ensure that their application is secure and is behaving as expected. P-T14's explained that the reason why they prefer to incorporate security in this stage was that it is more cost efficient to address security issues early in the SDLC. He explained, "*We have a small company, so it's very hard to catch all the bugs after release.*"

Developers test for security fortuitously. (SA) In other cases, security is not completely dismissed, yet it is not an explicit priority. Some security adopters run existing test suites that may include security at varying degrees. These test suites include test cases that any application is expected to pass, however, there is not necessarily a differentiation between security and non-security tests. Some developers run these tests because they are required to, without actual knowledge of their purpose. For example, P-T3 presumes that since his company did not have security breaches, security must be incorporated in existing test suites. He explained, "*[Security] has to be there because basically, if it wasn't, then our company would have lots of problems.*"

Developers' security testing is feature-driven. (SI) In another example where security is not dismissed, yet not prioritized, one participant from the security inattentive group (out of the only two who perform security testing), considers that security is not a concern as his application is not outward facing, *i.e.*, it does not involve direct user interaction. P-T9 explained, "*Security testing [pause] I would say less than 5%. Because we're doing embedded systems, so security [is] pretty low in this kind of work.*" While this may have been true in the past, the IoT is increasingly connecting embedded systems to the Internet and attacks against these systems are increasing [28]. Moreover, classifying embedded systems as relatively low-risk is particularly interesting as it echoes what Schneier [46] described as a road towards "a security disaster". On the other hand, P-T4 explained that only features that are classified as sensitive in the design stage are tested, due to the shortage in security expertise. As the company's only developer with security background, these features are assigned to P-T4. Other developers in T4 do not have security experience, thus they do not security-test their code and they are not expected to.

4.1.4 Code analysis stage

Eight developers reported that their teams have a mandatory code analysis stage. Participants from the security adopters group mentioned that the main objectives in this stage is to verify the code's conformity to standards and in-house rules, as well as detect security issues. On the other hand, participants from the security inattentive group generally do not perform this stage, and rarely for security.

Security is a priority during code analysis. (SA) All security adopters who perform this stage reported that security is a main component of code analysis in their team. T5 mandates analysis using multiple commercial tools and in-house tools before the code is passed to the next stage. T3 has an in-house tool that automates the process of analysis to help developers with the burden of security. P-T3 explained, "*[Our tool] automatically does a lot of that for us, which is nice, it does static analysis, things like that and won't even let the code compile if there are certain requirements that are not met.*" One of the advantages of automating security analysis is that security is off-loaded to the tools; P-T3 explains that security "*sort of comes for free*".

Security is a secondary objective during code analysis. (SI) P-T2 explained that in his team, developers' main objective when using a SAT is to verify conformity to industry standards. Although they might check security warnings, other security testing methods are considered more powerful. P-T2 explained, "*[SAT name] doesn't really look at the whole picture. [...] In terms of: is it similar to a security*

vulnerability testing? No. Pen testers? No. It's very weak." In addition to the lack of trust in SATs' ability to identify security issues, and similar to previous research (*e.g.*, [31]), our participants complained about the overwhelming number false positives and irrelevant warnings.

Developers rarely perform code analysis, never for security. (SI) Code analysis is not commonly part of the development process for the security inattentive group. According to their developers, T2, T6, and T15 use SATs, but not for security. Code analysis is performed as a preliminary step to optimize code and ensure readability before the code review stage, with no consideration to security.

Reasons for underusing SATs were explored in other contexts [31]. The two main reasons in our interviews were that their use was not mandated or that developers were unaware of their existence. We found that **Developers vary in awareness of analysis tools.** (SI) In addition to those unaware, some developers use SATs without fully understanding their functionality. P-T10 does not use such tools since it is not mandated and his teammates are unlikely to do so. He said, "*I know that there's tools out there that can scan your code to see if there's any vulnerability risks [...] We are not running anything like that and I don't see these guys doing that. I don't really trust them to run any kind of source code scanners or anything like that. I know I'm certainly not going to.*" Despite his awareness of the potential benefits, he is basically saying *no one else is doing it, so why should I?* Since it is not mandatory or common practice, running and analyzing SATs reports would add to the developer's workload without recognition for his efforts.

4.1.5 Code review stage

Most security adopters say that security is a primary component in this stage. Reviewers examine the code to verify functionality and to look for potential security vulnerabilities. P-T14 explained, "*We usually look for common mistakes or bad practices that may induce attack vectors for hackers such as, not clearing buffers after they've been used. On top of that, it's also [about the] efficiency of the code.*"

Contrarily, the security inattentive discount security in this stage—security is either not considered, or is considered in an informal and adhoc way and by unqualified reviewers. Code review can be as simple as a sanity check, or a walkthrough, where developers explain their code to other developers in their team. Some reviewers are thorough, while others consider reviews a secondary task, and are more inclined to accept the code and return to their own tasks. P-T10 explained, "*Sometimes they just accept the code because maybe they are busy and they don't want to sit around and criticize or critically think through everything.*" Moreover, reviewers in T9 examine vulnerabilities to assess their impact on performance. P-T9 explained, "*[Security in code review is] minimum, I'd say less than 5%. So, yeah you might have like buffer overflow, but then for us, that's more of the stability than security issue.*" We grouped participants' descriptions of the code review stage into four distinct approaches.

Code review is a formal process that includes security. (SA) All security adopters mentioned that their teams include security in this stage. For some teams, it is a structured process informed by security activities in previous stages. For example, security-related warnings flagged during the code analysis phase are re-examined during code reviews. Reviewers can be senior developers, or an independent team. Being independent, reviewers bring in a new perspective, without being influenced by prior knowledge, such as expected user input. P-T5 said, "*We do require that all the code goes through a security code review that's disconnected from the developing team, so that they're not suffered by that burden of knowledge of 'no one will do this', uh, they will.*" Sometimes reviewers might not have adequate knowledge of the applications. In such cases, T1 requires developers to explain the requirements and their implementation to the reviewers. P-T1 said, "*You have to explain what you have done and why. [...] so that they need not invest so much time to understand what is the problem [...] Then they will do a comparative study and they will take some time to go over every line and think whether it is required or not, or can it be done in some other way.*" Although cooperation between different teams is a healthy attitude, there might be a risk of developers influencing the reviewers by their explanation. P-T13 indicated the possibility of creating a bias when reviewers are walked-through the code rather than looking at it with a fresh set of eyes. He said, "*umm, I have not really thought about [the possibility of influencing the reviewers.] [...] Maybe. Maybe there is a bit.*"

Preliminary code review is done as a checkpoint before the formal review. (SA) This is an interesting example of developers collaborating with reviewers. P-T1 mentioned that reviewers sometimes quickly inspect the code prior to the formal review process and in case of a potential issue, they provide the developer with specific testing to do before the code proceeds to the review stage. This saves reviewers time and effort during the formal code review, and it could help focus the formal process on intricate issues, rather than being overwhelmed with simple ones.

Security is not considered during code review. (SI) The majority of the security inattentive participants explained that their teams' main focus for code review is assessing code efficiency and style, and verifying how well new features fulfill functional requirements and fit within the rest of the application. In fact, some participants indicated that their teams pay no attention to security during this stage. It is either not the reviewers' responsibility, or is not an overall priority for the team. P-T7 explained that because reviewers are developers, they are not required to focus on security. In addition to not being mandated, our participants explained that most developers in their teams do not have the necessary expertise to comment on security. P-T7 said, "*Probably in the two years that I've been working, I never got feedback [on] the security of my code [...] [Developers] don't pay attention to the security aspect and they can't basically make a comment about the security of your code.*"

Security consideration in code review is minimal. (SI) According to developers from the security inattentive group, some of their teams pay little attention to security during code review only by looking for obvious vulnerabilities. Additionally, this may only be performed if the feature is security-sensitive. In either case, teams do not have a formal method or plan, and reviewers do not necessarily have the expertise to identify vulnerabilities [22]. Our participants explained that reviewers are either assigned or chosen

by the developer, based on the reviewer's qualifications and familiarity with the application. However, this can have serious implications, *e.g.*, those who have security expertise will carry the burden of security reviews in addition to their regular development tasks. P-T12 explained that this caused the individuals who had security knowledge to become "*overloaded*". Although our data does not allow us to make such explorations, it is important to investigate the effect of workload on the quality of code reviews, and whether it has an effect on developers' willingness to gain security knowledge. For example, does being the person designated to do security code reviews motivate developers to gain security knowledge? Or would they rather avoid being assigned extra reviewing workload?

4.1.6 Post-development testing stage

Security is a priority during post-development testing. (SA) Three participants from the security adopters group mentioned that their project teams have their own testers that evaluate different aspects, including security. The general expectation is that the testers would have some security knowledge. Additionally, P-T12 mentioned that his company hires external security consultants for further security testing of their applications. However, because the testing process by such experts is usually "*more expensive and more thorough,*" (P-T12), they usually postpone this step until just before releasing the application. We identified two distinct motivations for performing security testing at this stage: **Post-development testing is used to discover security vulnerabilities, or for final verification.** (SA) Unsurprisingly, the majority of security adopters rely on post-development testing as an additional opportunity to identify and discover security vulnerabilities before their applications are put out to production. T1, on the other hand, expects security post-development testing to reveal zero vulnerabilities. P-T1 explained, "*If they find a security issue, then you will be in trouble. Everybody will be at your back, and you have to fix it as soon as possible.*" Thus, this stage is used as a final verification that security practices in the previous stages were indeed successful in producing a vulnerability-free application.

Similar to the code review stage, we found evidence of collaboration between the development and the testing team, however, **Testers have the final approval.** (SA). Testers would usually discuss with developers to verify that they understand the requirements properly, since they do not have the same familiarity with the application as developers. However, P-T5 explained that although developers can challenge the testing team's analysis, they cannot dismiss their comments without justification. Addressing security issues is consistently a priority. P-T5 said, "*[The testing team will] talk to the development teams and say, 'here's what we think of this', and the development team will sometimes point out and say, 'oh, you missed this section over here' [...] but one of the things is, we don't let the development teams just say, 'oh, you can't do that because we don't want you to'. So the security teams can do whatever they want.*" Cooperation between developers and testers could help clear ambiguities or misunderstandings. In T5 testers have some privilege over developers; issues raised by testers have to be addressed by developers, either by solving them or justifying why they can be ignored. P-T5 hinted that disagreements may arise between different teams, but did not detail how they are resolved. Further exploration of this subject is needed, taking into consideration the level of security knowledge of the development team compared to the testing team.

Security is prioritized in post-development testing for all of our security adopters, where they rely on an independent team to test the application as a whole. On the other hand, although post-development testing appears to be common to all teams from the security inattentive group (with the exception of T10), it often focuses primarily on functionality, performance and quality analysis, with little to no regard for security. Our analysis revealed the following insights and approaches to post-development security testing.

Security is not considered in post-development testing. (SI) According to their developers, two teams (T10, T15) do not consider security during this stage. T10 does not perform any testing, security or otherwise. The company to which T15 belongs has its own Quality Analysis (QA) team, though they do not perform security testing. P-T15 said, "*I've never seen a bug related to security raised by QA.*" The case of T15 is particularly concerning; many teams rely on this stage to address software security, while T15 does not. According to our data, security is not part of the development lifecycle in T15. It would be interesting to further explore why some teams completely ignore software security, and what factors could encourage them to adopt a security initiative.

Post-development testing plans include a security dimension. (SI) As mentioned earlier, P-T2 relies mainly on this stage for security testing, In addition, P-T6, and P-T13 say that their teams consider security during this stage. However, there seems to be a disconnect between developers and testers in T6; developers are unaware of the testing process and consider security testing out of scope. Despite her knowledge that security is included in this stage, P-T6 mentioned, "*I don't remember any tester coming back and telling [me] there are [any] kinds of vulnerability issues.*" T13 started integrating security in their post-development testing after a newly hired tester who decided to approach the application from a different perspective discovered a serious security issue. P-T13 explained, "*No one had really been thinking about looking at the product from security standpoint and so the new tester we had hired, he really went at it from 'how can I really break this thing?' [..] and found quite a few problems with the product that way.*" The starting point of security testing in T13 was a matter of chance. When an actual security issue was discovered in their code, security was brought to the surface and post-development testing started addressing security.

Through our analysis, we found that along the security prioritization spectrum, there are cases where security in this stage is driven by different factors, as explained below.

Some participants discussed that their team relies on a single person to handle security, thus security consideration is driven by specific factors. For example, in T4, **Post-development security testing is feature-driven.** (SI). P-T4 is the only developer in his company with security expertise, thus he is responsible for security. He explained that his company has limited resources and few employees, thus they focus their security testing efforts only on security-

sensitive features (*e.g.*, authentication processes), as flagged by the developers. Thus, the question is how reliable are assessments in this case given that they are done by developers with limited security expertise? On the other hand, in T7, **Post-development security testing is adhoc.** (SI) . P-T7 explained that they rely on a single operations-level engineer who maintains the IT infrastructure and handles security testing. Thus, testing is unplanned and could happen whenever the engineer has time or "*whenever he decides.*" P-T7 erroneously [50] presumes their applications are risk-free since they are a "*small company*", and thus they are not an interesting target for cyberattacks. Company size was used by some of our participants to justify their practices in multiple instances. Although in our data we did not find evidence to support that company size affects actual security practices, it shows our participants' perception.

We also found that an external mandate to the company can be a driving factor for security consideration. For example, P-T8 reported that his company needs to comply with certain security standards, thus his team performs security testing when they are expecting an external audit "*to make sure the auditors can't find any issue during the penetration test.*" In this case, **Post-development security testing is externally-driven.** (SI) Such external pressure by an overseeing entity was described as "*the main*" driving factor to schedule security testing; P-T8 explained that if it were not for these audits, his team would not have bothered with security tests. Mandating security thus proved to be effective in encouraging security practices in a team that was not proactively considering it.

As evidenced by our data, the security inattentive group's security practices, if existent, are generally informal, unstructured, and not necessarily performed by those qualified. The main focus is delivering features to customers; security is not necessarily a priority unless triggered, *e.g.*, by experiencing a security breach or expecting an external audit.

4.2 The adopters vs. the inattentive

In general, security practices appear to be encouraged in teams to which the security adopters belong. In contrast, as explained by participants from the security inattentive group, their teams' main priority is functionality; security is an afterthought. Contrary to a trend towards labelling developers as "the weakest link" [27], our analysis highlights that poor security practices is a rather complex problem that extends beyond the developer. Just as we have identified instances where developers lack security knowledge or lack motivation to address security, we have also identified instances where security was ignored or dismissed by developers' supervisors, despite the developer's expertise and interest. It is especially concerning when security is dismissed by those high in the company hierarchy. As an extreme case, P-T15 reported zero security practices in their SDLC; she explained "*To be honest, I don't think anybody cares about [security]. I've never heard or seen people talk about security at work [...] I did ask about this to my managers, but they just said 'well, that's how the company is. Security is not something we focus on right now.'*"

It was interesting to find that all our participants who identified themselves as developers of web applications and services, *i.e.*, in their current daily duties, (namely, P-T4, P-T6, P-T7, P-T8, P-T10, P-T13, P-T15) fall in the security inattentive group. Specific reasons for this are unclear. It may be because web-development is generally less mature and has a quick pace [44], and teams are eager to roll-out functionality to beat their competitors. In such cases, functional requirements may be prioritized and security may be viewed as something that can be addressed as an update, essentially gambling that attackers will miss any vulnerabilities in the intervening time. Teams who have not yet become victims may view this as a reasonable strategy, especially since patching generally does not requires end-user involvement (*e.g.*, web server fixes do not require users to update their software), making it a less complicated process. However, since participants building other types of software also fall in the security inattentive group, it is hard to draw a generic conclusion that web-development is particularly insecure.

Table 2 summarizes the themes that emerged from our analysis. As expected, we found conflicting themes between the security adopters and the security inattentive group, where the more secure themes consistently belongs to the security adopters. However, our analysis also revealed common themes (see Table 2), some of which are promising while others are problematic for security. On the positive side, participants from both groups discussed developers' role in security during implementation. On the other hand, participants from both groups also indicated a lack of attention to security in the design stage. Reasons leading to these common themes sometimes vary. Consider the theme *Developers do not test for security*; the security inattentive group ignored security testing because developers often lack the knowledge necessary to perform this task. Whereas for the security adopters, the reason is that security testing is not included in developers' tasks even if they have the required knowledge. In Section 6.2 we discuss factors that we identified as influential to security practices.

5. INITIATIVES AND BEST PRACTICES

After exploring real life security practices, how do these compare to security best practices? To answer, we offer background on popular sources of best practices. We then amalgamate them into a concise list of the most common recommendations. In Section 6, we discuss the relationship between practices found in our study and best practices.

5.1 Secure SDLC initiatives

This section gives a brief background on prominent processes and recommendations for secure software development.

Security Development Lifecycle (SDL). Microsoft SDL [8] is the first initiative to encourage the integration of security in the SDLC from the early stages. It consists of 16 security practices and can be employed regardless of the platform.

Building Security In Maturity Model (BSIMM). Currently maintained by Cigital [2], the BSIMM [6] recommends 12 main security practices. It provides high-level insights to help companies plan their secure SDLC initiative and assess their security practices compared to other organizations.

Open Web Application Security Project (OWASP) initiatives. OWASP's Software Assurance Maturity Model (SAMM) [3] recognizes 4 main classes of SDLC activities and provides 3 security best practices for each. Additionally, the Developer Guide [1] provides best practices for architects

Table 2: Summary of themes emerging from the security adopters and the security inattentive, and common themes between the two groups. Although common themes exist, driving factors for these themes may differ. See Section 4.2 for more details.

Security Adopters Themes	Common Themes	Security Inattentive Themes
	Design	
· Security design is very important	· Security is not considered in the design stage	· Security consideration in the design stage is adhoc
	Implementation	
	· Security is a priority during implementation · Developers' awareness of security is expected when implementing	· Security is not a priority during implementation · Developers take security for granted · Developers misuse frameworks · Developers lack security knowledge · Developers perceive their security knowledge inaccurately · Vulnerability discovery can motivate security
	Developer Testing	
· Developers test for security fortuitously · Security is a priority during developer testing	· Developers do not test for security	· Developers' security testing is feature-driven
	Code Analysis	
· Security is a priority during code analysis		· Security is a secondary objective during code analysis · Developers rarely perform code analysis, never for security · Developers vary in awareness of analysis tools
	Code Review	
· Code review is a formal process that includes security · Preliminary code review is done as a checkpoint before the formal review		· Security is not considered during code review · Security consideration in code review is minimal
	Post-development Testing	
· Security is a priority during post-development testing · Post-development testing is used to discover security vulnerabilities, or for final verification · Testers have the final approval		· Security is not considered in post-development testing · Post-development testing plans include a security dimension · Post-development security testing is feature-driven · Post-development security testing is adhoc · Post-development security testing is externally-driven

and developers, whereas the Testing Guide [4] focuses on best practices for testing and evaluating security activities.

Others. Additional resources for security best practices include: NASA's *Software Assurance Guidebook* [39], NIST's *Special Publication 800-64* [32], US-CERT's *Top 10 Secure Coding Practices* [47], as well as various articles emphasizing the importance of secure development [7, 36, 37, 57].

5.2 Security Best Practices

Available resources for security best practices vary in their organization and their presentation style, *e.g.*, they vary in technical details. Practitioners may find difficulty deciding on best practices to follow and establishing processes within their organizations [38, 42, 54]. To help frame security practices we identified, we collected recommendations from the sources discussed in Section 5.1 to compose a concise set of best practices. This resulted in an initial set of 57 unorganized recommendations varying in format and technical details. We then grouped related recommendations, organized them in high-level themes, and iterated this process to finally produce the following 12 best practices. Other amalgamations may be possible, but we found this list helpful to interpret our study results. The list could be of independent interest to complementary research in this area.

B1 **Identify security requirements.** Identify security requirements for your application during the initial planning stages. The security of the application throughout its different stages should be evaluated based on its compliance with security requirements.

B2 **Design for security.** Aim for simple designs because the likelihood of implementation errors increases with design complexity. Architect and design your software to implement security policies and comply with security principles such as: secure defaults, default deny, fail safe, and the principle of least privilege.

B3 **Perform threat modelling.** Use threat modelling to analyze potential threats to your application. The result of threat modelling should inform security practices in the different SDLC stages, *e.g.*, for creating test plans.

B4 **Perform secure implementation.** Adopt secure coding standards for the programming language you use, *e.g.*, validate input and sanitize data sent to other systems, and avoid using unsafe or deprecated functions.

B5 **Use approved tools and analyze third-party tools' security.** Only use approved tools, APIs, and frameworks or those evaluated for security and effectiveness.

B6 **Include security in testing.** Integrate security testing in functional test plans to reduce redundancy.

B7 **Perform code analysis.** Leverage automated tools such as SATs to detect vulnerabilities like buffer overflows and improper user input validation.

B8 **Perform code review for security.** Include security in code reviews and look for common programming errors that can lead to security vulnerabilities.

B9 **Perform post-development testing.** Identify security issues further by using a combination of methods, *e.g.*, dynamic analysis, penetration testing, or hiring external security reviewers to bring in a new perspective.

B10 **Apply defense in depth.** Build security in all stages of the SDLC, so that if a vulnerability is missed in one stage, there is a chance to eliminate it through practices implemented in the remaining stages.

B11 **Recognize that defense is a shared responsibility.** Address software security as a collective responsibility of all SDLC entities, *e.g.*, developers, testers, and designers.

B12 **Apply security to all applications.** Secure low risk applications and high risk ones. The suggested effort spent on security can be derived from assessing the value of assets and the risks, however, security should not be ignored in even the lowest risk applications.

6. INTERPRETATION OF RESULTS

In this section, we compare security practices from our study to best practices, present factors influencing those practices, and discuss future research directions. We comment on teams' practices as described by their developers (our participants), recognizing that we have only one perspective per team. Compliance (or lack thereof) to all best practices is not proof of a secure (or insecure) SDLC. However, this list of widely agreed upon best practices allows us to make preliminary deduction on the software security status quo.

6.1 Current practices versus best practices

Our analysis showed different approaches to security and varying degrees of compliance with best practices. The best practice with most compliance is B9; almost all participants reported that their team performs security post-development testing (to varying degrees). Contrarily, most do not *apply defense in depth* (B10); the security adopters do not consistently integrate security throughout the SDLC and the security inattentive group relies mainly on specific stages to verify security (*e.g.*, post-development testing). In addition, security is generally not a part of the company culture for the security inattentive group and they commonly delegate a specific person or team to be solely responsible for security. This leads to adhoc processes and violates B11: *recognize that defense is a shared responsibility*. Moreover, the security inattentive group violate B12 by ignoring security in applications considered low-risk without evidence that they performed proper risk analysis.

Deviations from best practices are apparent even from the design stage. The majority of participants indicate that their teams do not address security during design, contradicting B1–B3. Some developers may even deliberately violate the *Design for security* best practice (B2) to achieve their business goals and avoid extra work. On the other hand, the two participants who discussed formal consideration of security in design claim the advantages of having more informed development processes, identifying all relevant threats and vulnerabilities, and not getting distracted by irrelevant ones [47].

The implementation stage is particularly interesting; it shows the contradictions between the security adopters and the security inattentive. Participants from both groups *perform secure implementation* (B4), yet this only applied to three security inattentive participants. For most of the security inattentive group, *security is not a priority* and *developers take security for granted*, assuming that frameworks will handle security. While frameworks have security benefits [52], each has its own secure usage recommendations (*e.g.*, [5]), often buried in their documentations, and it is unclear if developers follow them. In fact, our study suggests that *developers misuse frameworks* by circumventing correct usage to more easily achieve their functional goals, another violation of B4. Moreover, despite their reliance on frameworks, participants report that security is not factored in their teams' framework choices (violating B5).

We found non-compliance with best practices in other development stages as well. For example, some teams do not include security in their functional testing plans, violating B6, and some teams do not perform code analysis, violating B7. Ignoring code analysis is a missed opportunity for automatic code quality analysis and detection of common programming errors [17]. Participants who said their teams use security code analysis tools, do so to focus subsequent development stages on the more unusual security issues. Others do not review their code for security (violating B8); rather code review is mainly functionality-focused. In some cases, participants said that reviewers do not have the expertise to conduct security reviews, in others they maybe overloaded with tasks, and sometimes code review plans simply do not include security.

6.2 Factors affecting security practices

Through close inspection of our results and being immersed in participants' reported experiences, we recognized factors that appear to shape their practices and that may not be adequately considered by best practices. We present each factor and its conflict with best practices, if applicable.

Division of labour. Best practices conflict with some of our teams' division of labour styles. Participants explained that some teams violate the *Apply defense in depth* (B10) best practice because applying security in each SDLC stage conflicts with their team members' roles and responsibilities. In some teams, developers are responsible for the functional aspect (*i.e.*, implementation and functional testing) and testers handle security testing. These teams are also violating B6, because integrating security in functional testing plans would conflict with the developers' assigned tasks. Complying with these best practices likely means they need to change the team's structure and re-distribute the assigned responsibilities. Teams may be reluctant to make such changes [42] that may conflict with their software development methodologies [35], especially since security is not their primary objective [27].

Security knowledge. We found that the expectation of security knowledge (or lack thereof) directly affects the degree of security integration in developers' tasks. When security knowledge was expected, participants said that developers were assigned security tasks (*e.g.*, performing security testing). On the other hand, we found that developers' (expected) lack of security knowledge resulted in lax security practices (*Security is not considered in the design stage*, *Security is not a priority during implementation*, *Developers do not test for security*, and *Security is not considered during code review*). While these violate best practices (*e.g.*, B1, B4 B6, B8), it is unrealistic to rely on developers to perform security tasks while lacking the expertise. From teams' perspective, they are relieving developers from the security burden. This may be a reasonable approach, loosely following recommendations of taking the developer out of the security loop when possible [14, 27]. Another obvious, yet complicated, answer would be to educate developers [8]. However, companies may lack the resources to offer security training, and there is evidence that developers remain focused mainly

on their primary functional task and not security [22].

Company culture. Another influential factor indicated by participants is the teams' cognizance of security and whether it is part of the company culture. In teams where security was reportedly advocated, developers spoke of security as a shared responsibility (conforming with B11). In instances where security was dismissed, participants said that developers did not consider security, and even those with security knowledge were reluctant to apply it. For successful adoption of security, initiatives should emerge from upper management and security should be rooted in the company's policies and culture. Developers are more likely to follow security practices if mandated by their company and its policies [59]. Integrating and rewarding security in the company culture can significantly motivate security practices [58, 59], compared to instances where security is being viewed as something that only "heroes" do if there is time.

Resource availability. Some participants said their team decides their security practices based on the available budget and/or employees who can perform security tasks. As reported, some teams violate B10 as they do not have enough employees who can perform all the recommended security tasks in addition to their original workload. Also, others reportedly violate B9, because they neither have the budget to hire external penetration testers, nor do their members have the expertise to perform such post-development tests. For such companies, the price for conforming with these best practice is too steep for little perceived gain. In other cases, participants said their team strains their resources in ways that can be detrimental. For example, the one developer with the most security knowledge is handed responsibility to identify security-sensitive features and to verify the security of the team's code. This is a significant burden, yet with little support or guidance. Besides the obvious security risks of such an approach, it may also lead to employee fatigue and ultimately to the loss of valuable team members.

External pressure. Monitoring by an overseeing entity can drive teams to adopt security practices to ensure they comply with its standards. Encouraging security practices through external mandates is not new, *e.g.*, the UK government mandated that applications for the central government should be tested using the National Technical Authority for Information Assurance CHECK scheme [11]. As a result of this initiative, companies have improved their management and response to cyber threats [10]. It would be interesting to explore how to mandate security practices in companies, and how governments and not-for-profit agencies could support teams, particularly those from the security inattentive group, to become more secure.

Experiencing a security incident. Participants reported that discovering a vulnerability or experiencing a security breach first-hand is another factor that encouraged security practices and awareness in their teams. Despite extensive publicity around security vulnerabilities, awareness of and commitment to security remains low [45]. Our analysis shows that direct vulnerability discovery influenced security practices more than hearing news-coverage of high-profile vulnerabilities (*e.g.*, [21, 53]). This can be explained by the optimistic bias [55]: the belief that "misfortune will not strike me" [45]. Rhee *et al.* [45] found that the optimistic bias strongly influences perception of security risks in Information Technology (IT). It is even greater when the misfortune seems distant, without a close comparison target. Thus, to overcome such bias, security training and awareness has to reach all levels–from upper management to those directly involved in the development process. Similar to Harbach and Smith's [29] personalized privacy warnings which led users to make more privacy-aware decisions, software security training should be personalized and provide concrete examples of the consequences of these threats to the company. We recommend that training should also not focus exclusively on threats; it should provide concrete proactive steps with expected outcomes. Additionally, it should include case studies and first-hand accounts of security incidents, and approaches to overcome them. Hence, security training moves from the theoretical world to the real world, aiding in avoiding the optimism bias.

6.3 Future research directions

Security best practices advocate for integrating security starting from the early SDLC stages. However, with limited resources and expertise, if a team can only address security in post-development testing, is this team insecure? Or might this testing be sufficient? Is the security inattentive group in our dataset really guilty of being insecure? Or did they just find the cost of following security best practices too steep? Available best practices fail to discuss the baseline for ensuring security, or how to choose which best practices to follow based on limited resources and expertise. It was also interesting to find that most security best practices are from industry sources and are not necessarily empirically verified.

For future research, we suggest devising a lightweight version of security best practices and evaluating its benefit for teams that do not have enough resources to implement security throughout the SDLC, or when implementing traditional security practices would be too disruptive to their workflow. Additionally, teams that succeeded at building a security-oriented culture should be further explored to better understand how others can adopt their approach. Further exploration of how to incorporate security in the company culture and evaluating its benefits can be a starting point for more coherent security processes, since developers are more likely to follow security practices if mandated by their company and its policy [59]. Particularly, what lessons can be carried from the security adopters over to the security inattentive group? Our work explores some of the issues surrounding secure development practices. Surveys with a larger sample of companies and more stakeholders would be an interesting next step.

7. CONCLUSION

Through interviews with developers, we investigated SDLC practices relating to software security. Our analysis showed that real-life security practices differ markedly from best practices identified in the literature. Best practices are often ignored, simply since compliance would increase the burden on the team; in their view, teams are making a reasonable cost-benefit trade-off. Rather than blaming developers, our analysis shows that the problem extends up in company hierarchies. Our results highlight the need for new, lightweight best practices that take into account the realities and pressures of development. This may include additional automation or rethinking of secure programming practices to ease the burden on humans without sacrificing security.

8. ACKNOWLEDGMENTS

We thank our participants for their time. H. Assal acknowledges her NSERC Postgraduate Scholarship (PGS-D). S. Chiasson acknowledges funding from NSERC for her Canada Research Chair and Discovery grants.

9. REFERENCES

[1] https://www.owasp.org/index.php/Category:OWASP_Guide_Project.

[2] https://www.cigital.com.

[3] www.owasp.org/index.php/OWASP_SAMM_Project.

[4] www.owasp.org/index.php/OWASP_Testing_Project.

[5] AngularJS Developer Guide. https://docs.angularjs.org/guide/security.

[6] BSIMM. https://www.bsimm.com.

[7] Cybersecurity Engineering. https://www.cert.org/cybersecurity-engineering/.

[8] Security Development Lifecycle. https://www.microsoft.com/en-us/sdl.

[9] *Content analysis for the social sciences and humanities*. Addison-Wesley Publishing Co., 1969.

[10] Cyber security boost for UK firms. https://www.gov.uk/government/news/cyber-security-boost-for-uk-firms, 2015.

[11] IT Health Check (ITHC): supporting guidance. https://www.gov.uk/government/publications/it-health-check-ithc-supporting-guidance/it-health-check-ithc-supporting-guidance, 2015.

[12] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the usability of cryptographic apis. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*, 2017.

[13] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *IEEE Symp. on Security and Privacy*, 2016.

[14] Y. Acar, S. Fahl, and M. L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8, Nov 2016.

[15] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl. Developers need support, too: A survey of security advice for software developers. In *Cybersecurity Development (SecDev), 2017 IEEE*, pages 22–26. IEEE, 2017.

[16] H. Assal, S. Chiasson, and R. Biddle. Cesar: Visual representation of source code vulnerabilities. In *VizSec'16*, pages 1–8, Oct.

[17] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh. Using static analysis to find bugs. *IEEE Software*, 25(5):22–29, Sept 2008.

[18] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi. Efficient and flexible discovery of php application vulnerabilities. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 334–349, April 2017.

[19] G. Berisha and J. Shiroka Pula. Defining small and medium enterprises: a critical review. *Academic Journal of Business, Administration, Law and Social Sciences*, 1, 2015.

[20] B. Chess and G. McGraw. Static Analysis for Security. *IEEE Security & Privacy*, 2(6):76–79, 2004.

[21] Codenomicon. The Heartbleed Bug. http://heartbleed.com.

[22] D. Oliveira *et al.* It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. In *ACSAC '14*, pages 296–305. ACM, 2014.

[23] S. Elo and H. Kyngäs. The qualitative content analysis process. *Journal of Advanced Nursing*, 62(1):107–115, 2008.

[24] A. Forward and T. C. Lethbridge. A taxonomy of software types to facilitate search and evidence-based software engineering. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, CASCON '08, pages 14:179–14:191, New York, NY, USA, 2008. ACM.

[25] D. Geer. Are Companies Actually Using Secure Development Life Cycles? *Computer*, 43(6):12–16, June 2010.

[26] B. G. Glaser and A. L. Strauss. *The discovery of grounded theory: strategies for qualitative research*. Aldine, 1967.

[27] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security Privacy*, 14(5):40–46, Sept 2016.

[28] A. Greenberg. Hackers Remotely Kill a Jeep on the Highway—With Me in It. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, 2015.

[29] M. Harbach, M. Hettig, S. Weber, and M. Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 2647–2656, New York, NY, USA, 2014. ACM.

[30] M. Howard and S. Lipner. *The security development lifecycle: SDL, a process for developing demonstrably more secure software*. Microsoft Press, Redmond, Wash, 2006.

[31] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *ICSE*, 2013.

[32] R. Kissel, K. Stine, M. Scholl, H. Rossman, J. Fahlsing, and J. Gulick. *Security considerations in the system development life cycle*. 2008.

[33] K. Krippendorff. Estimating the reliability, systematic error and random error of interval data. *Educational and Psychological Measurement*, 30(1):61–70, 1970.

[34] K. Krippendorff. Testing the reliability of content analysis data. *The content analysis reader*, pages 350–357, 2009.

[35] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[36] G. McGraw. *Software security: building security in*. Addison-Wesley, Upper Saddle River, NJ, 2006.

[37] G. McGraw. Seven myths of software security best practices. http://searchsecurity.techtarget.com/opinion/McGraw-Seven-myths-of-software-security-best-practices, 2015.

[38] P. Morrison. A Security Practices Evaluation Framework. In *Proceedings of the 37th International*

Conference on Software Engineering, ICSE '15, pages 935–938, Piscataway, NJ, USA, 2015. IEEE Press.

[39] NASA. Software Assurance Guidebook, NASA-GB-A201. https://www.hq.nasa.gov/office/codeq/doctree/nasa_gb_a201.pdf, 2002.

[40] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl. A stitch in time: Supporting android developers in writingsecure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1065–1077, New York, NY, USA, 2017. ACM.

[41] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 426–437, New York, NY, USA, 2015. ACM.

[42] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda. Can security become a routine?: A study of organizational change in an agile software development group. In *ACM CSCW*, 2017.

[43] J. Radcliffe. Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System. https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf.

[44] J. Ratner. Human factors and Web development, 2003.

[45] H.-S. Rhee, Y. U. Ryu, and C.-T. Kim. Unrealistic optimism on information security management. *Computers & Security*, 31(2):221 – 232, 2012.

[46] B. Schneier. Security Risks of Embedded Systems. https://www.schneier.com/blog/archives/2014/01/security_risks_9.html.

[47] R. Seacord. Top 10 secure coding practices. https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices, 2011.

[48] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *ESEC/FSE 2015*, pages 248–259. ACM, 2015.

[49] I. Sommerville. *Software engineering*. Pearson, Boston, 2011.

[50] J. Sophy. 43 Percent of Cyber Attacks Target Small Business. https://smallbiztrends.com/2016/04/cyber-attacks-target-small-business.html, 2016.

[51] C. Stransky, Y. Acar, D. C. Nguyen, D. Wermke, D. Kim, E. M. Redmiles, M. Backes, S. Garfinkel, M. L. Mazurek, and S. Fahl. Lessons learned from using an online platform to conduct large-scale, online controlled security experiments with software developers. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*, Vancouver, BC, 2017. USENIX Association.

[52] S. Streichsbier. Improve Web Application Security with Frameworks: A case study. http://www.vantagepoint.sg/blog/18-improve-web-application-security-with-frameworks-a-case-study.

[53] Symantec Security Response. ShellShock: All you need to know about the Bash Bug vulnerability. http://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability, 2014.

[54] I. A. Tondel, M. G. Jaatun, and P. H. Meland. Security Requirements for the Rest of Us: A Survey. *IEEE Software*, 25(1):20–27, Jan 2008.

[55] N. D. Weinstein and W. M. Klein. Unrealistic optimism: Present and future. *Journal of Social and Clinical Psychology*, 15(1):1–8, 2017/08/12 1996.

[56] J. Witschey, S. Xiao, and E. Murphy-Hill. Technical and personal factors influencing developers' adoption of security tools. In *ACM SIW*, 2014.

[57] C. Woody. Strengthening Ties Between Process and Security. https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/strengthening-ties-between-process-and-security#touch, 2013.

[58] G. Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, NSPW '08, pages 89–97, New York, NY, USA, 2008. ACM.

[59] S. Xiao, J. Witschey, and E. Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *ACM CSCW*, 2014.

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

APPENDIX

A. INTERVIEW SCRIPT

The following questions represent the main themes discussed during the interviews. We may have probed for more details depending on participants' responses.

- What type of development do you do?
- What are your main priorities when doing development? (In order of priority)
- Do your priorities change when a deadline approaches?
- What about security? Is it something you worry about?
- How does security fit in your priorities?
- Which software security best practices are you familiar with?
- Are there any obligations by your supervisor/employer for performing security testing?
- What methods do you use to try to ensure the security of applications?
- Do you perform testing on your (or someone else's) applications/code?
- Do you perform code reviews?

B. PARTICIPANTS DEMOGRAPHICS

Table 3: Participants demographics

	Participant					Company and team	
Participant ID	Gender	Age	Years	Title	SK	Company size	Team size[1]
P-T1	F	30	1	Software engineer	4	Large enterprise	20
P-T2	M	34	15	Software engineer	5	Large enterprise	12
P-T3	M	33	10	Software engineer	4	Large enterprise	10
P-T4	M	38	21	Software developer	4	SME	7
P-T5	M	34	12	Product manager	5	Large enterprise	7
P-T6	F	26	3	Software engineering analyst	3	Large enterprise	12
P-T7, P-T8*	M	33	4	Senior web engineer	4	SME – n/a*	3
P-T9	M	34	5	Software developer	3	Large enterprise	20
P-T10, P-T11*	M	33	8	Software engineer	2	SME – SME*	5
P-T12	M	37	20	Principal software engineer	5	SME	10
P-T13	M	38	15	Senior software developer	2	SME	8
P-T14	M	26	3	Software developer	2	SME	4
P-T15	F	27	5	Junior software developer	4	Large enterprise	7

Years: years of experience in development
SK: self-rating of security knowledge 1(no knowledge) - 5(expert)
$\star$: indicates participant's previous company
SME: Small-Medium Enterprise
[1] Team size for the current company

C. DEGREE OF SECURITY IN THE SDLC

Table 4: Extending Table 1 to show the degree of security in the SDLC and the application type. ● : secure, ○ : somewhat secure, × : not secure, ⊗ : not performed, ? : no data

(a) The Security Adopters

Application		Design	Implementation	Developer testing	Code analysis	Code review	Post-dev testing
embedded software	T1	×	●	×	●	●	●
design and engineering software	T3	?	●	?	●	●	●
design and engineering software	T5	●	●	○	●	●	●
info. management & decision support	T11	?	●	○	●	●	?
support utilities	T12	×	●	○	●	●	●
support utilities	T14	×	●	●	⊗	●	●

(b) The Security Inattentive

Application		Design	Implementation	Developer testing	Code analysis	Code review	Post-dev testing
kernels	T2	×	●	×	○	○	●
website content management	T4	○	○	○	⊗	○	○
e-finance	T6	×	○	×	×	○	○
online productivity	T7	×	×	×	⊗	×	○
social networking	T8	×	×	×	⊗	×	●
embedded software	T9	●	●	○	⊗	○	○
online booking	T10	○	○	×	⊗	○	⊗
online productivity	T13	×	●	×	⊗	○	●
online productivity	T15	×	×	×	×	×	×