Security During Application Development: an Application Security Expert Perspective

Tyler W Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford
University of North Carolina at Charlotte
Department of Software and Information Systems
tthoma81@uncc.edu, mtabassu@uncc.edu, billchu@uncc.edu, Heather.Lipford@uncc.edu

ABSTRACT
Many of the security problems that people face today, such as security breaches and data theft, are caused by security vulnerabilities in application source code. Thus, there is a need to understand and improve the experiences of those who can prevent such vulnerabilities in the first place - software developers as well as application security experts. Several studies have examined developers' perceptions and behaviors regarding security vulnerabilities, demonstrating the challenges they face in performing secure programming and utilizing tools for vulnerability detection. We expand upon this work by focusing on those primarily responsible for application security - security auditors. In an interview study of 32 application security experts, we examine their views on application security processes, their workflows, and their interactions with developers in order to further inform the design of tools and processes to improve application security.

ACM Classification Keywords
H.2.m Information Interfaces and Presentation (e.g. HCI): Miscellaneous; D.2.4 Software Engineering: Software/Program Verification, Testing and Debugging, Management

Author Keywords
Application Security Experts; Security Vulnerabilities; Secure Programming; Software Development

INTRODUCTION
A large number of security issues are caused by a type of software bug, known as a security vulnerability. These vulnerabilities can be extremely damaging when exploited. The infamous 2017 Equifax breach was the direct result of a web application security vulnerability [5, 17]. As of this writing, this breach is believed to have compromised detailed personal information of one hundred forty three million people, or almost half of the population of the United States [5, 17]. According to the United States Department of Homeland Security, 6,449 new software vulnerabilities were *reported* in 2016 alone [8], and the number of actual vulnerabilities in the software we use is likely far higher than this. Security vulnerabilities can be exploited through a variety of attacks, such as cross site scripting and SQL injection, to steal private information, harming both individuals and organizations. Thus, to the extent possible, security vulnerabilities should be found and remediated in application source code.

Security vulnerabilities are introduced by developers while creating software. Organizations implement a variety of processes to find and fix vulnerabilities in the software development lifecycle, such as security audits and static analysis. There are a large number of commercial tools, as well as numerous research projects aimed at helping detect such security issues.

Yet, developers are not heavily involved in such security processes [10, 16]. Prior research suggests that developers feel that security is the responsibility of other parties, and consequently avoid engaging in security practices when possible, delegating to other people and processes within the organization [10, 16, 31]. Additional research has examined why developers have difficulty using static analysis tools to detect security vulnerabilities, including large numbers of false positives, lack of collaboration support, and complicated tool output [10]. Other researchers are examining guidelines and approaches for improving the usability of tools for developers [16, 19, 20].

All of this existing and ongoing work has focused on the perspectives and needs of developers. We expand upon these results by focusing on the people who currently do application security work. Application security is often the responsibility of a "Software Security Group," or SSG, within organizations. Application security experts within that group are charged with performing static and dynamic analysis to find security vulnerabilities in application source code. By focusing on this important group of users, we examine additional needs and experiences of all people involved in application security work to further inform the design of tools and processes for reducing security vulnerabilities in applications.

In this paper, we report the results of an interview study of application security experts who *examine source code* for security issues, who we refer to as security auditors. Our goals are to gain an understanding of the security processes in their

CHI 2018, April 21–26, 2018, Montreal, QC, Canada
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-5620-6/18/04. . . $15.00
DOI: https://doi.org/10.1145/3173574.3173836

organizations, their interactions with developers and other stakeholders, and their perceptions on the challenges they face and impact on application security. This knowledge will support the research and development efforts of tools aimed at these security experts, and the developers they work with. The end goal is to improve the experiences and interactions of developers and security experts, thereby improving the security of applications, and reducing the likelihood and occurrences of security attacks on applications.

RELATED WORK

Prior work has explored both the needs and behaviors of security experts, as well as more specific behaviors and perceptions of security within software development.

Security experts

A variety of research has examined the behaviors, perspectives, and needs of security experts, identifying a range of common challenges including a lack of security training in organizations, communication issues between security experts and other stakeholders, and the lack of efficient security tools [24, 25, 26]. Security experts must communicate with a variety of other stakeholders throughout an organization to resolve security issues. Thus, there is need for security tools to better support this collaboration and reduce communication overhead.

Previous research on security experts has also produced guidelines for security tools for general security experts. Such guidelines should also apply, at least to some degree, for application security tools for both auditors and developers. For example, Werlinger et al. recommended that security tools should decrease complexity and communication overhead, disseminate knowledge, and provide flexible reporting [25]. Similarly, guidelines from Jaferian at al. on security management tools include customizability, easy to change configurations, automatic detection of security issues, and different interfaces for different stakeholders [9]. We extend these results with more depth regarding the detailed issues within the domain of application security and vulnerability detection and mitigation.

Security in Software Development

Organizational factors, such as organizational structure, organizational politics, and managerial pressure, can impact the quality of the code that developers write [11, 12, 13]. The security of the code is similarly impacted. For example, Kroksch & Poller examined the potential of an external security consultation to lead to new security routines within an Agile development team [15]. They found that the intervention did lead to increased awareness and desire to incorporate security as an important quality of their software. Despite this interest, security was not ultimately made an integral part of the development process because such security work was discouraged by the organizational structure and organizational goals. This case study emphasizes the importance of not just increasing developers' attention to security issues, but also to the need for organizations to develop routines and reward the security work that developers do. Similarly, Jing et al. found "A disconnect between developers conceptual understanding of security and their attitudes regarding their personal responsibility and practices for software security [31]." Even with awareness of the importance of security, developers expected other people and processes to take care of those issues. Thus, simply increasing developers' knowledge of security vulnerabilities will not necessarily lead to an increase in their performing security work.

Researchers have also examined the challenges that developers face in correctly utilizing security APIs and security-related documentation and resources in their code. For example, Fahl et al. demonstrated the mistakes developers make in incorrectly or insecurely using SSL APIs [6], and simplified cryptographic APIs have been shown to be more usable [28]. A study of cryptographic APIs also demonstrated that documentation is crucial for both usability and security [1]. One difficulty is that formal API documentation may be secure, but can be difficult to use. Formal security guidance provided to developers may also not provide sufficiently detailed solutions [1, 3]. Instead, developers turn to informal but less secure sources such as StackOverflow [2, 3].

Developers and Security Analysis Tools

Researchers have established that developers underuse security tools, focusing in particular on static analysis tools, for a variety of reasons. Through an interview study of developers, Johnson et al. found that while developers' perceived static analysis tools to be beneficial, they did not use them due to high false positive rates, and the ways in which warnings were presented to users [10]. Developers are also impacted by their social and organizational environment, and are more likely to adopt security tools when their peers already use and trust those tools [27]. Interacting with security experts can also increase developers' feelings of responsibility for securing their code [27].

However, commercial static and dynamic analysis tools may not be sufficiently usable for developers, who will have more limited security knowledge [18]. Smith et al. examined the kinds of questions that developers ask when assessing possible security vulnerabilities found by static analysis tools [20]. In categorizing these questions, they found that developers do a lot more work to understand a vulnerability and its remediation than simply reading the warning notification. Developers need support for a wide range of code understanding, including tracing the flow of untrusted data, comparing vulnerabilities against previous examples, and finding additional documentation. Tools do not adequately help developers resolve vulnerabilities, resulting in failures to efficiently and successfully address the vulnerability warnings [19].

Other researchers have created new static analysis tools to address the specific needs of developers. At Google, Sadowski created the static analysis tool Tricorder, with the specific aim of increasing developer use of static analysis methods by decreasing false positive warnings [16]. Developers submitted their own static analyzers, which were incorporated into the tool only if user feedback indicated sufficiently low uninteresting warnings. Do et al. propose a just-in-time approach through layered static analysis, providing developers with the most relevant warnings alongside their development activities in real time [4]. The ASIDE tool introduced interactive static analysis, integrating security vulnerability warnings and mitigations alongside the code that developers write [21, 30]. Similarly, the FixDroid tool provides Android developers with warnings and quick fixes covering a variety of common security pitfalls [14]. Developers have been shown to appreciate in-context security feedback and easy mitigation options [4, 14, 29, 32]. None of these previous efforts have examined vulnerability detection and mitigation within context of organizational teams and processes for developing and releasing software. Thus, we aim to add to these results by examining the viewpoint of an additional stakeholder, the application security expert.

METHODOLOGY

We conducted an interview study of experts in application security, seeking participants who either examine source code or perform static or dynamic analyses. For simplicity, we refer to these experts as security auditors throughout the paper. We recruited these experts using a snowball sampling technique, utilizing contacts from a variety of sources and then asking participants to recommend additional experts. We started with our own personal contacts, and made additional contacts at two security events, namely the CyberSecurity Symposium at our university held annually for regional security professionals as well as OWASP's 13th Annual AppSecUSA Security Conference.

We recruited a total of thirty-two participants from twenty-seven different organizations, see Table 1. Job titles varied greatly, and included "Security Consultant" (n=4), "Security Engineer" (n=5), and "Systems" or "Software" Engineer with a security role (n=5). As P6 stated, *"In the industry there's security analyst, there's security engineer, appsec engineer, appsec analyst, there's a lot of names. But they should all be able to do the same thing."* Several participants held senior positions, with titles containing "Manager" (n=2), "Director" (n=6), or "Vice President" (n=2). Participants reported an average of 10.72 years (SD 5.51) of professional experience as a security expert. Twenty-seven participants had previously been employed as a developer, with an average of 9.22 (SD 6.24) years of professional experience. All of our participants were male, and 25 Caucasian. While many organizations did not produce commercial software, all participants discussed working at least in part on customer-facing software.

We conducted and recorded an interview over the phone with each participant. The interviews were semi-structured, with a set of basic questions that were varied depending on the participant's background and the answers they provided. We asked about participants' typical workday and responsibilities for software security. We then asked about their own as well as their organization's processes and tools for finding and mitigating vulnerabilities, how they interact with developers, the biggest challenges they and their organizations face in software security, and what solutions they think could address those challenges. Interviews ranged from 30 - 45 minutes, and experts were provided a $10 gift card as a thank you for participating. The study was approved by our university's IRB.

We collected our demographic data at the end of the interviews with a set of closed, structured questions. As part of this, we asked participants to rate their general security knowledge, secure programming knowledge, and programming skills on a scale of one to ten. Participants responded with an average of 8.13 (SD 1.34) for security knowledge and 8.23 (SD 1.23) for secure programming knowledge. Additionally, participants responded with an average of 6.61 (SD 1.37) for programming skills. As expected, participants rated themselves highly on their security knowledge, with the low standard deviation demonstrating that they felt similarly about these ratings.

Interviews were transcribed. We followed an inductive coding process, looking for both common patterns of response to the questions as well as interesting topics and comments. Two researchers independently and iteratively coded five sample participants, comparing and merging their code books with discussion between all authors. Agreement was reached on the codebook and all codes for those 5 participants, resulting in a codebook of 39 separate codes. The two coders then coded all remaining participants independently with no further changes to the codebook. When coding was complete, the researchers compared each individual code and discussed and resolved any disagreements. Disagreements were tracked, and inter-rater reliability was calculated at 95.15%. Codes were then grouped into higher level categories which form the subsections of the Results section below.

In addition to the coding, we examined all of the transcribed interviews to perform workflow analysis. Based on each participant's responses, we created a workflow model for each participant and their organization. Once all diagrams were complete, we analyzed all of the diagrams, looking for common patterns and trends. Following this, we iteratively formed an aggregate workflow model representing the most common elements and patterns. Lastly, we analyzed the differences between each of the workflow diagrams as a basis to discuss the variations on our common workflow model.

RESULTS

We begin by describing the general workflow and security processes our participants described. We then focus on the communication and organizational issues that impacted vulnerability detection and remediation. Following this, we briefly discuss technical challenges and needs that they encountered. Our results are all qualitative, but we report the number of participants with similar comments to highlight how prominent different views were in our sample.

Security Processes

Participants worked in organizations ranging from small software companies to very large financial institutions and technology companies, see Table 1. Although processes and workflows for finding and fixing vulnerabilities varied slightly between participants, most were quite similar. Figure 1 shows an aggregate workflow model that represents our participants and their organizations.

Table 1. Participant Demographics

ID	Job Title	Organization Type	ID	Job Title	Organization Type
P1	Principle Application Security Engineer	Large Business Management Software	P17	Senior Security Consultant	Small Consulting
P2	Information Security Engineer	Large Financial Institution	P18	Director of Information Security	Medium Security Software
P3	Director of Security Solutions	Large Internet Service Provider	P19	Systems Engineer (Security Role)	Medium Network Security Services
P4	Senior Software Engineer (Security Focus)	Large Engineering and Consulting	P20	Senior Security Engineer	Medium Call Center Analytics
P5	Technology Manager Information Security	Large Financial Institution	P21	Cloud and Application Security Manager	Large Consulting and Training
P6	Application Security Engineer	Large Healthcare Software	P22	Director of Security Strategy	Medium Risk Management
P7	Security Lead	Large Technology Infrastructure	P23	CTO (Chief Technology Officer)	Small Security Consulting
P8	Senior Security Architect and IT Architect	Large Hardware and Software	P24	Software Developer (Security Focus)	Small Network Security
P9	Sales Engineer	Medium Security Software	P25	Director of Engineering	Small Network Security
P10	Senior Vice President (Role: Security Architect)	Large Financial Institution	P26	Senior Technology Project Coordinator	Large Security Foundation
P11	Senior Security Director	Large Technology Infrastructure	P27	Principle Security Consultant	Medium Consulting
P12	Director of Security Research and Development	Large Security Software	P28	CTO (Chief Technology Officer), CoFounder	Medium Security Software
P13	Program Manager	Medium Consulting	P29	CSO (Chief Security Officer)	Small Security Software
P14	Application Security Consultant	Medium Security Software	P30	Software Engineer	Large Network Infrastructure
P15	Vice President of Threat Research Center	Medium Security Software	P31	Application Security Consultant	Medium Cyber Risk Management
P16	Senior Security Researcher	Medium Security Software	P32	Senior Application Security Engineer	Large Entertainment

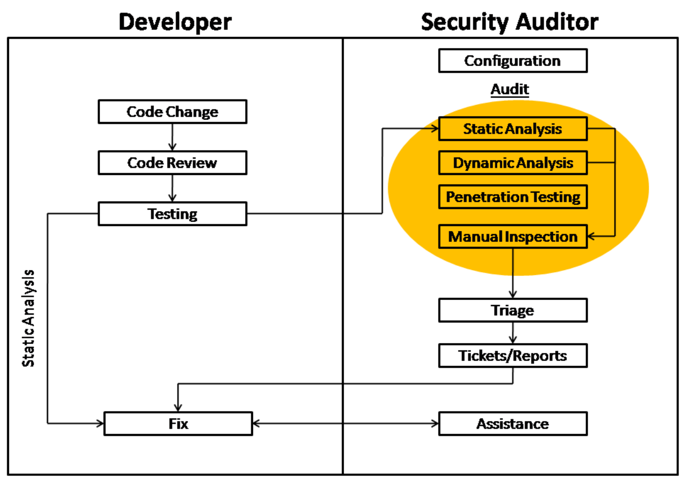

Figure 1. Aggregate Security Auditor Workflow Model

Whether the organization followed a more traditional waterfall-like process and SDLC (Systems Development LifeCycle), or an AGILE methodology, software developers would first perform their implementation tasks. Our auditors (n=25) did mention that code reviews and testing were regularly performed by developers looking for all kinds of bugs and quality issues, but that the auditors were not involved in that process as security was rarely a focus of them.

Our participants were all part of a separate security group operating outside of development teams. Auditors often oversaw the security of many different applications throughout the organization. Security processes were initiated after changes to the code were complete. Two timeline schedules were observed. If security processes were scheduled at regular intervals, such as annually, the security process was considered an audit. Audits continued to be performed on released code to find issues in the deployed software. Otherwise, the security processes would be scheduled based on the needs of the project and organization. Both timelines used the same processes of static analysis, dynamic analysis, pen testing, and manual inspection to find security vulnerabilities. More details of these processes are described below.

Once a vulnerability was detected, it was then triaged. This means that the order in which security bugs were remediated was prioritized based on their severity. Security vulnerabilities were often documented alongside all other bugs using bug tracking tools, the most popular of which was JIRA[1]. In other instances, the results of all security processes would be compiled in a separate, out of band report produced for the development team. Addressing vulnerabilities then becomes part of the typical bug fixing process undertaken by the development team. In other words, all participants reported that remediation of a vulnerability was always performed by a developer.

A number of participants discussed the use of continuous integration processes within their development teams. Amazon defines Continuous Integration as "A DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. [7]" When continuous integration was used, static analysis was done on the build server every time a new build was created and pushed to production. The results would be integrated into the bug tracking platform and be shown as security bugs. Depending on the severity of the issue, the bugs could fail the build. The developer would then have to fix these bugs and recommit the code. The "integrated" static analysis rules were always configured by a security auditor.

All configuration of security tools and processes was performed by security auditors themselves as needed. In addition to their primary role of finding security vulnerabilities, ten participants also mentioned that they participated in threat modelling or security architecture discussions as part of the overall software development process, but did not mention specifically when those activities occurred.

Finding Bugs

The primary responsibility of all of our participants was to find security vulnerabilities in application source code. As expected, all participants reported using static analysis and dynamic analysis tools, such as Fortify[2] and AppScan[3], to detect security vulnerabilities. The participants conducted all static and dynamic analyses themselves. Static and dynamic analysis were sometimes perceived differently within an organization and performed at separate times. For example, as P27 stated:

[1]https://jira.atlassian.com/
[2]https://saas.hpe.com/en-us/software/application-security
[3]http://www-03.ibm.com/software/products/en/appscan

"In my experience, organizations treat them (static and dynamic analysis) as separate steps and oftentimes I feel that's the result of the way security controls inside of guidance and policy documents express those tasks. They're not treated as a holistic set of controls. They aren't done as a combined activity and I feel that's a bit of a disservice to the practice of application security." -p27

In addition to static and dynamic analysis, twelve auditors mentioned the use of penetration testing, sometimes performed by a separate security team. Finally, two participants used bug bounties to uncover problems in released code that were otherwise missed. Not surprisingly, many (n=24) participants in our sample mentioned the management of false positives and false negatives as a challenge of all of these processes.

Thirty participants reported that they frequently examined code for vulnerabilities on their own, referring to this as "manual code inspection." This was done to understand discovered vulnerabilities, as part of a security audit, or to investigate issues such as authentication, datastore accesses, and architectural flaws that may not be found by other means. Inspections also occurred to help create customized rules for tools to reduce false positives:

"All applications end up needing custom rules written or some sort of tweak to make the (static or dynamic analysis) scanner work for that application. Manual code review is how you identify those patterns and make sure that your rules are covering them appropriately. We'd typically go through and we'd say 'Alright, let's go find the CSRF protection. Let's go find their encoding mechanism. How are they handling their database queries?"' -p17

However, code inspections were done on a limited scale due to the sheer volume of code.

Another major responsibility of our participants was performing regular audits of projects. Based on different risk designations, projects were required to undergo such audits over various intervals of time. For example, as P2 states:

"Basically there's three levels. That's high, medium, and low, and the high risk projects have to be reviewed every year. The medium ones are two years. The low ones, every three years." -p2

Static analysis, dynamic analysis, and sometimes audit meetings were conducted as a part of the audit process. Our participants described the meeting as similar to a traditional code review, however it involved security auditors, project leads, and sometimes developers.

"If a section of code is deemed very important and critical, then we do get assigned reviewers who have worked on security areas before and then it's like a peer review; they check it. It's normal code review process, but with a emphasis on... the security aspect of it." -p24

Participants would meet face to face and review code in order to identify code smells which may carry risk.

"So, the reviews are targeted typically based on two elements; one is that static analysis has told us that there is a problem or even dynamic analysis has told us there is a problem so they will look in that area. And secondly, where have code changes been made or where have new features been added to the code. So, those are high-risk spots and those are the ones that are targeted for inspection." -p7

Detected vulnerabilities were again triaged. However, verification that developers fixed the issues brought up during these audits was not performed until the next scheduled audit.

Fixing Bugs

Triage was an important aspect of our security auditors' workflow, reflecting their perceptions of risk. Serious vulnerabilities would be given a deadline for resolving. Vulnerabilities deemed critical, or higher risk, would have much shorter deadlines. The criteria by which vulnerabilities were triaged were often varied and proprietary. Generally, the probably of an exploit occurring and the impact if it did occur were the two primary factors in the triaging process.

"We're looking at things like, how it can be exploited, how many, how likely is it to be exploited, what are the assets that we're trying to protect? Because obviously if you have an application that's got only, let's say a thousand confidential records at risk, then that's a lot less of a risk to a company as if they're 100,000 confidential records, right? So those types of things are taken all into account. That is part of the risk analysis." -p2

While security auditors would follow through on the more serious vulnerabilities to ensure they were resolved, a project could still continue to move into production containing known vulnerabilities.

"And, unless that's critical or high severity in nature, a lot of lingering issues remain simply by the quantity of issues that come out" -p27

In order to assist developers, participants may provide static or dynamic analysis results, documentation, or suggestions within the bug tracking tool. However, release of the software can occur at any time, at the discretion of management. In most of our participants' organizations, it is possible for release to occur with known security vulnerabilities if management is willing to accept the risk.

"It'll be assigned to a developer. Typically we try to provide as much information in there so that they can reproduce the issue." -p21

Summary

To summarize the common security practices reported by our participants:

- Security auditors are solely responsible for detecting security vulnerabilities in code.
- Their processes are labor intensive and often manual.
- Responsibility for fixing vulnerabilities lies with development teams.

Developer Interaction

One of our goals was to examine the interaction that application security experts have with the developers who write the code. While our interviews confirmed that auditors are responsible for detecting bugs, fixing them required communication between auditors and developers as shown in the workflow model in Figure 1.

Communication Patterns

To understand communication patterns, we coded any time an auditor reported communication with someone else. Communication between security auditors and developers did not occur as frequently as we expected. Most communication revolved around how to remediate a vulnerability detected during static analysis. Thus, the communication involved notifying the developer of the issue and providing guidance for fixing it. Further interaction could be initiated by a security auditor with an interest in seeing a vulnerability resolved, or by a developer with questions about how to resolve a vulnerability.

Most auditors described communication primarily occurring through the bug tracking or ticketing system. Auditors would provide explanations on what the vulnerability is, and pointers on how to fix it. Developers may then make additional comments to ask questions or for more help. For more in depth conversations, developers would contact the auditor more directly. For example:

"So there is some interaction there but it's typically through the ticketing system where we'll trade some comments back and forth. There'll definitely be like a re-test. 'Can you make sure it works for me?' They'll ask. Sometimes if, like I said, if they're new to us, if they're new to security, they might ask for a ten minute call just to make sure that they understand the issue." -p21

As P21 stated, how communication occurred was based on the team and the individual developer. Auditors stated that they tried to engage developers wherever they were most comfortable interacting, whether that be in person, over email, or simply through the comments in the bug tracking system.

In addition to the developer assigned to remediate the vulnerability, auditors also interacted with other members of the development team. Analysis results were sometimes packaged as a report and provided to the project, rather than to an individual developer. In these cases, the project lead or a designated developer with an interest in security, known as a security champion, would interact with the security auditor. Our participants also interacted with developers as they sought to understand the security properties and use cases of an application during dynamic analysis, manual code inspection, or security code review.

Finally, security auditors interacted with developers during training. Many (n=23) of the security training programs were designed and taught by the auditors themselves. Training could be general application security training, or customized to the types of errors that developers are currently making.

"What we often do is periodically assess the category of vulnerabilities, what the kinds of volume and mistakes the developers are making and provide a very small part of training just on those things." -p22

While training has obvious goals of increasing the knowledge of developers, participants also commented on the importance of raising developer awareness of the auditors themselves:

"While we can argue about whether secure code training gets us anywhere, at least it raises the visibility into these issues and people come back. Now whether or not they remember, you know, all the different things that are out there, probably not. But as long as they know how to access you and they know that you're there as a resource, it is certainly very beneficial." -p21

Communication Challenges

The importance of communication to fixing bugs means that communication breakdowns can impact the security of the application when bugs are not properly addressed. One primary communication challenge was coordinating between different teams. Developers and security teams operated independently, and some organizations even had separate teams for static and dynamic analysis. Auditors also oversaw applications for many different teams of developers throughout the organization. Naturally, this leads to communication bottlenecks as auditors must navigate the different cultures and needs of these different teams to coordinate the reporting and fixing of security vulnerabilities.

To make matters worse, security auditors must sometimes coordinate with different teams of developers and project stakeholders to get approval to fix issues. Divergent interests of developers, managers, and other stakeholder means that this is sometimes very difficult. Many participants (n=10) commented that such communication issues could cause delays in security problems being fixed.

"There's this approval process, IT has concerns, marketing has concerns, and you need that unanimous approval and that if you don't get it that vulnerability persists." -p3

Participants perceived that much of their role is in motivating and convincing other parties to implement security solutions. Security auditors in our study struggled with convincing developers or other stakeholders that a security issue was real and in need of remediation. Many (n=9) participants mentioned that developers had difficulty in seeing a harmless example exploit and understanding that the vulnerability was serious. For example, one participant mentioned that developers frequently did not understand that alert boxes delivered through cross site scripting vulnerabilities could be easily replaced with credential stealing, false login forms by an attacker.

"The biggest question they have is 'Why, why do I actually need to do this, why do I care?' They usually call me out on that, and say, 'So how do you weaponize this, right? What's the attack path here? I don't see an actual vulnerability. All I see is some alert box"' -p17

Auditors also mentioned that communicating vulnerabilities found outside of static analysis could be challenging. Static analysis tools report a particular line of code potentially containing a bug. This means that notification and subsequent remediation suggestions can be provided in the context of the source code. However, dynamic analysis is performed by using test data in a running application. Therefore, if an issue is found, the lines of code causing this issue must be first determined before they can be fixed. Several participants (n=3) mentioned that it was very difficult for developers to interpret the results of dynamic analysis scans, locate the problem in the code, and then apply the correct fix. Similarly, two participants mentioned that developers struggled to understand how security problems in the architecture of applications were actual security issues. In these cases, auditors had to leverage knowledge of the functionality of applications to demonstrate architectural or logic flaws.

Another commonly reported issue was determining the best way to communicate and motivate different groups of people. For example, three participants commented on the need to be sensitive to the developers' feelings about their code when notifying them of security issues in that code.

"Obviously, if you're examining one of the applications that they have written, you just have to be very sensitive to the fact that... we call it 'calling someone's baby ugly.' So you have to, if you're going to point out a security flaw, you have to do it in a way that's respectful." -p11

Overall, this means that auditors must be familiar with other stakeholders' knowledge and needs in order to determine how to best communicate with them regarding security issues. P14 talked about the importance of this message delivery.

"I think a lot of what I deal with is not necessarily the technology, it's the how you communicate it effectively to developers. My boss likes to joke that we are 60 percent psychologists and 40 percent security professionals. The hard part is determining for the organization that we're talking to at that moment, what is the best way of communicating the solution and the risk associated with an app in a way that's going to resonate with developers and cause them action, and also not make them freak out." -p14

Security Champions

One key type of developer that ten participants mentioned was someone they referred to as a security champion. The security champion was a member of the development team who was an advocate for security. This was an unofficial designation, with no formal security training needed. Instead, they only needed an interest in security.

"The security champion is typically a volunteer from the engineering team who is interested in security, and more importantly interested in reducing the incidents of security vulnerabilities. I work with him or her and discuss what the priorities should be for that particular product as we move forward towards the next release." -p7

As these participants stated, security champions served as important liaisons between the security group and development teams. They help to extend the efforts of the limited quantity of security auditors and can serve as an advocate for security on each development team. When security champions are involved, the security auditor can communicate with the security champion, and the security champion can then communicate with each of the fellow members of the development team. This requires less communication effort for the auditor, with fewer needs to motivate the champion regarding security. The security champion can then communicate as a peer to fellow developers. Consequently, six auditors mentioned that security champions were either required for the developer code review process, or frequently participated in it. Auditors (n=3) also recognized the benefit to "train the trainer" and educate the security champion instead of all of the developers. The hope is that the security champion could then train other developers on his/her development team. The downside is that other members of the development team may not receive such training and security reports directly, resulting in decreased security awareness overall.

Participants clearly tried to cultivate the relationships they maintained with security champions in a variety of ways. For example, while these roles were informal and not tied to pay, auditors mentioned rewarding champions with trips to security conferences to enhance their training and maintain their interest in security.

Summary

Communication with development teams and other stakeholders were important aspects of a security auditor's job, with auditors needing to motivate developers to understand and fix the security problems found. Challenges and breakdowns to this communication could result in vulnerabilities remaining in the application. Security champions served as important and valued liaisons between the auditors and development teams.

Organizational Challenges

Auditors discussed a variety of challenges faced within their organizations that impact how application security is performed and its effectiveness.

Balancing Risk and Resources

Twenty-two of our participants mentioned that balancing risk against resource limitations is a key security challenge. The first issue was understanding the risk of the various products, which can be difficult to characterize. In addition, as several (n=4) of the participants mentioned, not all products have the same level of risk.

"The risk level that's acceptable for product A isn't the same as it is for product B. So you don't treat everything the same, you can't just state across the board here's our watermark for risk for the organization." -p7

These risks must then be weighed against available resources, which are limited. As participants mentioned, security costs money and organizations must strike a balance between delivering a product and ensuring that the product is free of security issues. And application security is just one aspect of the security of an organization.

"If you look at percentage, for security in general, and particularly between what I would call traditional infrastructure and network security, versus application security. Those numbers are horrifically in favor of the more traditional network and operational security." -p26

These limitations mean that rarely does an organization do all possible activities for vulnerability detection and remediation.

"The reality is that most of the dev efforts will implement one or two of the things I've just described but not all of them. In each one of those areas, you get a different lens into the application. You get a different opportunity to identify and re-mediate security concerns. Without implementing them all, there's a chance, and this is what we see all the time, there's a chance a security vulnerability is being missed and making it through to the final product." -p22

One strategy participants attempted was to reduce their effort through automating tools and processes. More automation was also commonly stated (n=14) as a potential improvement to application security. Yet, participants also commented that many of the existing processes and tools do not lend themselves well to automation. In addition, auditors were also aware of the importance of human judgement in security decisions.

"A human just has kind of this intuition about where to look and how to basically defeat the system." -p30

Limited Security Expertise

A key resource limitation is the skilled and knowledgeable security experts themselves. For example, P10 estimated that there are roughly 50 open positions to each qualified person. Another stated:

"There's just not enough people that know security. So we can't even find the resources that we would need to actually get the job done." -p21

This results in very few application security experts. As P15 states, *"If you have ten employees you actually have a fairly large security team."* This also means that the ratio of application security experts to developers is also very small. Of the participants who reported concrete numbers, they all reported a single auditor for one to two hundred developers. Rather than rely solely on application security experts, auditors emphasized the importance of training developers in security.

However, only two of our participants mentioned that secure programming training was mandatory in their organizations, and as a result, according to P15 *"The percentage of people that do the training is very, very small."* Similarly, several lamented the lack of security education. *"We have to train developers at a university level. Computer science graduates, to this day, don't get secure code in training."* -p15

Fitting in Security

As described earlier, application security experts traditionally are organized as part of a separate software security group, overseeing almost all of the detection of security vulnerabilities. This also reflects how an application's security is viewed within development teams, as a non-functional requirement that is not built in from the beginning. As P10 states:

"I would say that the big challenge there is that security and development have traditionally been disjointed and they have been separate teams. Security is the watchdogs, development does the work and all security has ever done is scan stuff. So write code, ask questions later. And we have to change that" -p10

One reason for this lack of security integration may be that security was not an important feature for small or immature applications. Yet, over time, security has grown more important. Thus, organizations may not start out with strong security processes, which means they later need to determine how to add security on top of already complex software development processes. Participants acknowledged this issue:

"Where are we going to plug in threat modeling? Where are we going to plug in static scanning? Where are we going to plug in dynamic scanning? When are we doing pen-testing? How are we doing developer training? What's the process to fix bugs? How do we prioritize those bugs? Just getting that very well integrated into what was already a well-oiled machine is doable, but it's challenging." -p20

Summary

With security as just one of many competing requirements within software development, organizing and utilizing the very limited security resources and personnel has a major impact on the resulting security of the end product. Auditors themselves are limited resources, who must understand the risks of their software and balance those against available time and efforts to be effective.

Technical Challenges and Needs

Participants in our sample described many challenges and needs within the tools that they used to find vulnerabilities. These issues led to a significant amount of work to run these tools and utilize the results.

Scalability and Performance

As with other studies of experts [24, 25], many (n=13) participants mentioned scalability as a key technical challenge. The specific issues for application security tools include:

- Locating, acquiring, configuring, and using tools that can adequately support all relevant languages and frameworks of an organization;
- Time and effort required to write rules for static and dynamic analyzers;
- Managing large numbers of false positives from scanning large applications or many applications; and
- Modern static analysis and dynamic analysis tools are not sufficiently robust with large applications and have slow performance, yet it is very labor intensive to configure them to scan only pieces of an application.

Participants were most frustrated with the last issue above - that their tools may just not work at all or work too slowly to be useful. They stated that modern static and dynamic analyzers tend to crash when scanning large amounts of code or crawling large applications.

"I think all the current tools lack performance. Lack the ability to go fast. The reason why a lot of the testing tools don't fit into a traditional DevOps model is because they're really slow." -p28

"I've had a scan running for at least two or three days, I just had to kill it." -p12

Legacy and Complex Code

Participants frequently mentioned both legacy and complex code as key security challenges. In order to properly identify security vulnerabilities in code, it is often necessary for security auditors to obtain a limited understanding of a given application's functionality. If the person who wrote the code is no longer available, the security auditor must locate a suitable proxy. Unfortunately, they can be both difficult to locate and their knowledge may be incomplete. This makes the evaluation of false positives and the remediation of vulnerabilities much more difficult.

Third Party Libraries

Third party code was also very challenging (n=13) for two reasons. First, participants stated that vulnerabilities are very common in frameworks and libraries, even well-known established ones. Yet, determining the problem as well as actually fixing it may be beyond their abilities, requiring a patch from the library vendor. Additionally, third party libraries can break static analysis and greatly contribute to false positives. This is because entry into a 3rd party library or component, when the source code is not available, constitutes a "boundary" or a "breakage." This breaks the analyses path from the source to the sink and destroys the taint signature that is used by static analysis algorithms.

"As soon as you trace that into a third party component, its kind of a dead end there." -p12

Thus, tools may report many vulnerabilities simply because the code relies on a third party library.

Automation and Smarter Tools

Many participants expressed a desire for more automation or intelligence in security tools. For example, seven participants wanted tools which are capable of assisting in triaging and remediating vulnerabilities. While participants had varied wishlists, their desires often related to tools requiring less manual configuration to run, and enabling easier data sharing and analysis across tools. In other words, auditors wanted tools to relieve more of their mundane and tedious manual labor. They acknowledged that creating smarter or more automated tools was challenging, but they were hopeful that with time, tools would improve.

Limitations

Security experts can be difficult to recruit for user studies, which leads to the use of convenience sampling techniques that come with limitations. Our participants are not necessarily representative of all application security experts. Given our recruiting methods, we likely have more experts from large organizations, with larger security groups and established security processes. Still, the results demonstrate the problems even mature organizations encounter surrounding security. We also have primarily U.S. based organizations, and processes and security cultures may differ in other countries. Our sample is also all male, and while the security field is primarily male, we were disappointed to not have a more gender-representative sample.

IMPLICATIONS

As other researchers have already described, developers who write code are often not very involved in detecting security vulnerabilities in that code [10, 15, 31]. Additionally, given our auditors' complaints regarding the lack of security knowledge and training by developers, developers are likely not taking the necessary steps to prevent many of those vulnerabilities in the first place, even though they are ultimately tasked with fixing them. Placing sole responsibility for software security in the hands of very few application security experts has a profound impact, resulting in known vulnerabilities slipping into production, long delays, added expense to fix vulnerabilities, and insufficient analysis to detect potentially serious vulnerabilities. Thus we, like others, believe that these results point to the need to increase developer attention to preventing vulnerabilities in their code. Many of our participants directly advocated for greater developer involvement in the security process and the benefits of doing so:

"And that's the biggest challenge, getting security moving all the way to the left and addressing every single phase of the development life cycle....Then I guess 80 percent of the kind of low hanging fruit vulnerabilities like SQL Injection that we find out in the wild would be addressed." -p17

Preventing more vulnerabilities from being committed in the first place would also free up auditors to focus on more complex or difficult to detect vulnerabilities, to more deeply monitor and analyze the risk profiles of various projects, and to provide more in-depth assistance to development teams. Involving other stakeholders may also relieve auditors of the burden of motivating and coordinating vulnerability remediation.

Participants also suggested ways to increase developer involvement in vulnerability prevention. For example:

- Several auditors (n=3) suggested that the Agile process be modified to treat security as a functional requirement instead of "technical debt." This may shift the importance of security in the eyes of developers since it would be treated at the same level as other functionality.
- Other auditors (n=6) discussed increasing the use of continuous integration, where security bugs are entered automatically into defect tracking systems with other general bugs. Security bugs are more likely to be on a level playing field with other bugs if they are continuously found and potentially prevent code from being committed. However, as mentioned earlier, performance and scalability issues make it challenging to run scans quickly.
- Many of our participants (n=19) also called for more security training for developers to provide them with a greater awareness of security and the knowledge necessary to produce fewer security bugs.

Yet, as prior work has clearly demonstrated, the burden of security work needs to be lowered for developers and auditors alike [2, 3, 23, 28]. In addition, organizational processes need to support and reward that work rather than delegate it to one small group of security experts [15]. We believe our findings suggest three particular areas that need increased attention and future research to achieve these goals.

Security training. Auditors commented that improved training is needed for developers, such as training that is more engaging (one participant suggested gamification for example), and concrete, involving code the developer is actually working on. Auditors also felt it should be targeted towards the issues developers are actually encountering, and delivered in-situ, as developers encounter those issues in their code. Commercial tools do not currently do this. There were several concrete suggestions for training support, such as:

- (p4, p20, p31) Tools which run in the developers' IDE, that provide feedback directly to the developer about why something is wrong, and help generate code for developers;
- (p4) Tools that report problems to supervisors or others to determine the kinds of security training needed;
- (p22) Tools for collaboration around security vulnerabilities, providing contextualized communication and help beyond just the location of a bug.

We, and others, are working towards developing tools to achieve some of these goals [4, 21, 22, 30]. We are encouraged to find additional support from auditors of these design needs.

Increasing automation As shown both by existing research [10, 16] and our results, analysis tools produce many false positives, and is one reason that developers are discouraged from using such tools. Our application security experts also indicated that false positives were a major challenge for them as well. Experts are required to manually create custom rules and configuration for every application to reduce these false positives, which is time consuming.

"I would love to see a static analysis tool where I could get it out of the box, run it on my application and it just worked. No tuning required...But right now I had never found a tool that allows me to just go from zero to a good or a reasonable scan result. Just doesn't exist." -P28.

False positives can be reduced with more accurate analysis algorithms, and our participants pointed out particular pain points around third-party libraries for example. However, providing more automated support for the tuning process to help users customize security scanners could also be helpful and greatly reduce the burden of using static and dynamic analysis tools.

Additionally, utilizing continuous integration processes requires detection tools to run automatically. Yet, as our auditors mentioned, many tools do not yet have sufficient performance to be run in this way, or are simply too difficult to configure to do so. Tool builders need to investigate more incremental and layered mechanisms for scanning and reporting detected vulnerabilities in order to provide faster and more localized feedback. This would also support the more interactive analysis tools proposed by several researchers to provide developers with real-time and interactive security feedback [30, 14].

Risk assessment. Assessing and acting upon security risk is a key role of auditors. Yet, this process was primarily manual. Auditors considered risk during vulnerability triage, in determining audit frequency, and in choosing which parts of the code to spend more time on. There are currently few tools that provide auditors and other stakeholders with assessments of risk to an application to support decision making. Similarly, few tools help users prioritize issues found, or are configurable based upon risk assessments. As one participant commented:

"I don't have a good tool that lets me say, 'Hey, what's, you know, what's my risk posture right now? ... am I doing as well as I should be?"' -p20

Thus, tools need to reflect such risk assessments. For example, when should static analysis results fail a build and how should it be configured to do that? If builds are broken too frequently, organizations will incur a severe cost. On the other hand, if builds are not broken frequently enough, applications will be developed with serious security issues. How much effort should developers put toward fixing different vulnerabilities? Which vulnerabilities are the most serious and which are not? If developers perform more vulnerability prevention and detection themselves, their tools will also need to reflect such risk assessments to help prioritize and direct their limited amount of time towards addressing the most important security issues.

CONCLUSION

To our knowledge, this paper is the first user study to specifically focus on application security experts. Our results provide further evidence that application security work is primarily performed by these experts, separately from software development teams. Separating security from development adds communication overhead and barriers. And finding and fixing security vulnerabilities so late in the software lifecycle can result in costly delays and expense to applications, and an increased likelihood that applications are not adequately secure. In particular, our results highlight the importance of triage to reflect risk assessments, the challenges of security communication that mostly occurs within bug tracking systems, the experts' role of motivating developers to fix problems, the importance of security champions, and the desires for less configuration and more automation. These results further demonstrate that improving application security will involve a combination of organizational processes to incentivize and support developers in focusing on security issues earlier, developer training to give them more concrete and actionable knowledge and motivation, and tools for both developers and experts that reduce the manual burdens of configuration and analysis. We hope these results can inform a variety of process and tool improvements to reduce the costs of vulnerability detection and remediation, and allow application security experts to focus on deeper and more complex security issues and processes.

ACKNOWLEDGMENTS

We would like to thank the National Science Foundation (Grants NSF-DGE #1523041 and NSF-CNS #1318854) for supporting this work.

REFERENCES

1. Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. 154–171. DOI: http://dx.doi.org/10.1109/SP.2017.52
2. Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*. 289–305. DOI:http://dx.doi.org/10.1109/SP.2016.25
3. Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*. 22–26. DOI: http://dx.doi.org/10.1109/SecDev.2017.17
4. Lisa Nguyen Quang Do, Karim Ali, Benjamin Livshits, Eric Bodden, Justin Smith, and Emerson Murphy-Hill. 2017. Just-in-time Static Analysis. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 307–317. DOI: http://dx.doi.org/10.1145/3092703.3092705
5. Nathan Bomey Elizabeth Weise. 2017. Equifax data breach: Identity-theft hackers exploited flaw experts flagged in March https://www.usatoday.com/story/money/2017/09/15/equifax-data-breach-what-you-need-know-hacking-crisis/670166001/. (2017).
6. Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security (CCS '13)*. ACM, New York, NY, USA, 49–60. DOI: http://dx.doi.org/10.1145/2508859.2516655
7. What is Continuous Integration? https://aws.amazon.com/devops/continuous integration. 2017. (2017).
8. Department of Homeland Security National Institute of Standards and Technology National Vulnerability Database. https://web.nvd.nist.gov/view/vuln/statistics results. 2017. (2017).
9. Pooya Jaferian, David Botta, Fahimeh Raja, Kirstie Hawkey, and Konstantin Beznosov. 2008. Guidelines for Designing IT Security Management Tools. In *Proceedings of the 2Nd ACM Symposium on Computer Human Interaction for Management of Information Technology (CHiMiT '08)*. ACM, New York, NY, USA, Article 7, 10 pages. DOI: http://dx.doi.org/10.1145/1477973.1477983
10. Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 672–681. http://dl.acm.org/citation.cfm?id=2486788.2486877
11. M. Lavallée and P. N. Robillard. 2015. Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 677–687. DOI:http://dx.doi.org/10.1109/ICSE.2015.83
12. E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan. 2015. The Design Space of Bug Fixes and How Developers Navigate It. *IEEE Transactions on Software Engineering* 41, 1 (Jan 2015), 65–81. DOI: http://dx.doi.org/10.1109/TSE.2014.2357438
13. Nachiappan Nagappan, Brendan Murphy, and Victor Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*. ACM, New York, NY, USA, 521–530. DOI: http://dx.doi.org/10.1145/1368088.1368160
14. Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. 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)*. ACM, New York, NY, USA, 1065–1077. DOI: http://dx.doi.org/10.1145/3133956.3133977
15. Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '17)*. ACM, New York, NY, USA, 2489–2503. DOI:http://dx.doi.org/10.1145/2998181.2998191
16. Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 598–608. http://dl.acm.org/citation.cfm?id=2818754.2818828
17. Kanishka Singh. 2017. Equifax says web server vulnerability led to hack https://www.reuters.com/article/us-equifax-cyber/equifax-says-web-server-vulnerability-led-to-hack-idUSKCN1BP0CB. (2017).
18. Yannic Smeets. 2015. Improving the Adoption of Dynamic Web Security Vulnerability Scanners. In *Master's Thesis, Radboud University*. https://pdfs.semanticscholar.org/1981/583700ada13fa5fc376999726a3545c90891.pdf

19. Justin Smith. 2016. Identifying Successful Strategies for Resolving Static Analysis Notifications. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*. ACM, New York, NY, USA, 662–664. DOI: http://dx.doi.org/10.1145/2889160.2891034

20. Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions Developers Ask While Diagnosing Potential Security Vulnerabilities with Static Analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 248–259. DOI: http://dx.doi.org/10.1145/2786805.2786812

21. Tyler Thomas, Bill Chu, Heather Lipford, Justin Smith, and Emerson Murphy-Hill. 2015. A Study of Interactive Code Annotation for Access Control Vulnerabilities. In *Proceedings of the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '15)*. IEEE Computer Society, Washington, DC, USA.

22. Tyler W. Thomas, Heather Lipford, Bill Chu, Justin Smith, and Emerson Murphy-Hill. 2016. What Questions Remain? An Examination of How Developers Understand an Interactive Static Analysis Tool. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/thomas

23. Rodrigo Werlinger, Kirstie Hawkey, and Konstantin Beznosov. 2008. Security Practitioners in Context: Their Activities and Interactions. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems (CHI EA '08)*. ACM, New York, NY, USA, 3789–3794. DOI: http://dx.doi.org/10.1145/1358628.1358931

24. Rodrigo Werlinger, Kirstie Hawkey, and Konstantin Beznosov. 2009. An integrated view of human, organizational, and technological challenges of IT security management. *Information Management & Computer Security* 17, 1 (2009), 4–19. DOI: http://dx.doi.org/10.1108/09685220910944722

25. Rodrigo Werlinger, Kirstie Hawkey, Kasia Muldner, Pooya Jaferian, and Konstantin Beznosov. 2008. The Challenges of Using an Intrusion Detection System: Is It Worth the Effort?. In *Proceedings of the 4th Symposium on Usable Privacy and Security (SOUPS '08)*. ACM, New York, NY, USA, 107–118. DOI: http://dx.doi.org/10.1145/1408664.1408679

26. Rodrigo Werlinger, Kasia Muldner, Kirstie Hawkey, and Konstantin Beznosov. 2010. Preparation, detection, and analysis: the diagnostic work of IT security incident response. *Information Management & Computer Security* 18, 1 (2010), 26–42. DOI: http://dx.doi.org/10.1108/09685221011035241

27. Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying Developers' Adoption of Security Tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 260–271. DOI: http://dx.doi.org/10.1145/2786805.2786816

28. Glenn Wurster and P. C. van Oorschot. 2008. The Developer is the Enemy. In *Proceedings of the 2008 New Security Paradigms Workshop (NSPW '08)*. ACM, New York, NY, USA, 89–97. DOI: http://dx.doi.org/10.1145/1595676.1595691

29. Jing Xie, Bill Chu, Heather Richter Lipford, and John T. Melton. 2011a. ASIDE: IDE Support for Web Application Security. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*. ACM, New York, NY, USA, 267–276. DOI: http://dx.doi.org/10.1145/2076732.2076770

30. Jing Xie, Heather Lipford, and Bei-Tseng Chu. 2012. Evaluating Interactive Support for Secure Programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2707–2716. DOI: http://dx.doi.org/10.1145/2207676.2208665

31. J. Xie, H. R. Lipford, and B. Chu. 2011b. Why do programmers make security errors?. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 161–164. DOI: http://dx.doi.org/10.1109/VLHCC.2011.6070393

32. Jun Zhu, Jing Xie, Heather Richter Lipford, and Bill Chu. 2014. Supporting secure programming in web applications through interactive static analysis. *Journal of Advanced Research* 5, 4 (2014), 449–462.