# Building and Validating a Scale for Secure Software Development Self-Efficacy

**Daniel Votipka, Desiree Abrokwa, and Michelle L. Mazurek**
University of Maryland, College Park
dvotipka@cs.umd.edu, dabrokw1@umd.edu, and mmazurek@cs.umd.edu

## ABSTRACT
Security is an essential component of the software development lifecycle. Researchers and practitioners have developed educational interventions, guidelines, security analysis tools, and new APIs aimed at improving security. However, measuring any resulting improvement in secure development skill is challenging. As a proxy for skill, we propose to measure self-efficacy, which has been shown to correlate with skill in other contexts. Here, we present a validated scale measuring secure software-development self-efficacy (SSD-SES). We first reviewed popular secure-development frameworks and surveyed 22 secure-development experts to identify 58 unique tasks. Next, we asked 311 developers—over multiple rounds—to rate their skill at each task. We iteratively updated our questions to ensure they were easily understandable, showed adequate variance between participants, and demonstrated reliability. Our final 15-item scale contains two sub-scales measuring belief in ability to perform vulnerability identification and mitigation as well as security communications tasks.

## Author Keywords
Secure Development; Scale Development

## INTRODUCTION
Software developers play a critical role in end-user security, but secure development can be difficult. According to NIST's National Initiative for Cybersecurity Education (NICE) framework, developers must consider 44 distinct areas of security tasks [73]. However, many developers do not believe they have this level of skill [4], and prior work shows developers struggle to write secure code [76, 54, 2, 1]. Unfortunately, commonly used secure-development skill measures, e.g., the number of vulnerabilities reported in a developer's code, their scores on simple assessment exercises, or self-reported experience, can be noisy or time consuming to collect. This is problematic, as an accurate and efficient measure of skill is needed both to test whether various security education [57, 18, 43, 51, 86] and guidelines [73, 83, 20, 64, 68] are effective, and to act as a covariate in studies of new security tools [54, 90, 100, 94, 104, 87, 50, 89], documentation [2], and APIs [1, 48, 103].

We propose an alternative approach to measuring secure-development skill: a validated scale. Human behavior researchers regularly develop scales to "measure elusive phenomena that cannot be observed directly" due to cost or complexity [7]. Specifically, we propose measuring developers' secure-development self-efficacy—belief in one's ability to successfully perform a task— which correlates with actual skill in other contexts [7]. This scale would measure developers' belief in their ability to complete secure-development tasks, such as identifying security problems during software design or employing secure programming languages.

In this paper, we develop and evaluate a new scale for assessing developers' belief in their ability to perform tasks necessary to produce secure software: the Secure Software Development Self-Efficacy Scale (SSD-SES). We followed Netemeyer et al.'s 4-step scale creation process [72]:

1. **Construct Definition and Content Domain:** Clearly identifying the targeted construct's scope (i.e., the underlying idea). We focus on tasks related to secure code production.
2. **Generating and Judging Measurement Items:** Defining an initial pool of candidate scale questions (*items*) and ensuring they are relevant to the construct and understandable by respondents. We generated items by reviewing five popular secure-development frameworks. We judge questions based on reviews from security experts and developers.
3. **Developing and Refining the Scale:** Using Exploratory Factor Analysis (EFA) to identify an underlying factor structure (i.e., any sub-components of the targeted construct and their associated questions). We also refine the item set to its most efficient form (i.e., only including items with sufficient variance among respondents), while maintaining reliability (i.e., whether the scale consistently measures the construct).
4. **Finalizing the Scale:** We use Confirmatory Factor Analysis (CFA) to confirm the previously identified underlying factor structure holds, maintaining reliability, with a new sample.

All our procedures were approved by our institution's experimental-ethics organization. Throughout this process, our scale was evaluated with 311 software developers and 22 security experts. The final SSD-SES consists of 15 items measuring two underlying factors: vulnerability discovery and

*CHI'20,* April 25–30, 2020, Honolulu, HI, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-6708-0/20/04...$15.00

DOI: https://doi.org/10.1145/3313831.XXXXXXX

mitigation, and security communication. We show that SSD-SES performs reliably over multiple samples and behaves as expected with respect to relevant measures from prior work.

## RELATED WORK

Several previous studies sought to measure secure development skills to evaluate educational interventions [86, 22, 102, 30, 23, 85, 29, 11, 37, 49, 56, 21, 78] or study developer behaviors [76, 86, 1, 70, 71, 69]. In most cases, this has taken the form of having participants identify and exploit vulnerabilities in sample programs or write small programs with security-critical functionality. For example, while studying security API misuses, Oliveira et al. asked participants to identify common mistakes in security "puzzles" (i.e., code snippets) [76]. In a more intensive evaluation, Ruef et al. asked their Build It, Break It, Fix It competitors to write medium size programs with several security requirements during a week-long "build" round [86]. Participants then evaluate each others' submissions through vulnerability-demonstrating exploits in another week-long "break" round. These assessments provide valuable insights into actual secure-development skill, but are very cumbersome. Our work provides a more light-weight alternative.

There have been several efforts to develop scales for efficiently measuring end-user security and security expertise. To measure end-user security behaviors, Egelman and Peer created the Security Behavior Intentions Scale (SeBIS) [31] and Faklaris et al. established a measure for Security Attitudes (SA-6) [34]. Together these scales cover participants' security thoughts and behaviors; however, due to the difference in domains, we expect our scale measures an orthogonal construct.

Also related to our work are Rajivan et al.'s [81] and Woon and Kankanhalli's [105] security expertise scales. Rajivan et al.'s measure assesses security expertise by asking participants if they have performed several network defense and system administration tasks (e.g., configuring a firewall) along with two open-ended questions asking participants to describe security concepts (i.e., certificates and phishing). While this scale targets expert users, it again measures an orthogonal domain (e.g., network defense and system administration).

Woon and Kankanhalli's secure-development intentions scale focuses on development and includes self-efficacy questions [105]. However, it is a much broader measure, intended to assess several factors influencing secure-development practice adoption. Therefore, their self-efficacy questions are limited and less concrete than ours (e.g., "I would feel comfortable carrying out secure development of applications on my own"). By focusing specifically on self-efficacy, we can provide a more precise measure and identify underlying factors. Because this scale has received only preliminary validation, we did not use it to establish discriminant validity.

## ITEM GENERATION AND JUDGMENT

The first step in scale development is *construct definition*: scoping what the scale will and won't cover. As SSD-SES's goal is to measure software developers' belief in their ability to perform secure development tasks, we focus only on tasks related to the production of secure code. That is, we do not include tasks from parts of the software development lifecycle such as deployment, maintenance, or monitoring. We also restrict our tasks to those prescribed by widely accepted secure-development frameworks or experienced security experts.

### Initial Item Generation

The second step is generating a set of candidate *items* (questions). The goal is to thoroughly survey the construct domain and build an extensive possible item pool [72], to be narrowed in later steps. We chose initial items by analyzing four popular secure-development frameworks: NIST's National Initiative for Cybersecurity Education (NICE) framework [73], the Building Security In Maturity Model (BSIMM) [64], the Open Web Application Security Project (OWASP) Software Assurance Maturity Model (OSAMM) [20], and Microsoft's Security Development Lifecycle (SDL) [68].

Two researchers independently reviewed each framework, identified a set of prescribed tasks, and selected tasks focused on secure code production. The researchers then met to combine lists. Because best practice recommends a conservative approach to initial item generation (i.e., including any possibly related items) [72], if a task was identified by either researcher it was included in the initial set. Finally, the researchers merged tasks from different frameworks considered identical (e.g., phrased differently or using different terminology, but expressing the same idea), again conservatively.

This process produced 57 unique software-development-specific tasks mentioned in at least one framework. The full task set (with sources) is shown in Table 1. These tasks can be divided into six categories: determining security requirements (A1-11), identifying attack vectors (A12-14), identifying vulnerabilities (A15-22), implementing mitigations to prevent or remedy vulnerabilities (A23-37), testing of security requirements (A38-42), and effectively communicating about security with peers, leadership, and security experts (A43-57).

### Content Review

To ensure the identified tasks cover the full range of the domain (*content validity*) and that the task wording was understandable to software developers (*face validity*), we surveyed 22 secure-software-development experts and 8 developers.

**Expert review.** We asked security experts to review our initial 57 tasks (Table 1) and rate them on a 4-point Likert-scale ranging from *Definitely not a secure development task* to *Definitely a secure development task*. Respondents also had an *Unsure* option if the wording was confusing or they could not clearly delineate the task's appropriateness. We also asked respondents whether the task phrasing was unclear or confusing, and to explain any confusion in free text. We concluded by asking experts to list any missing tasks. Our expert review survey text is given in our supplementary material.

We recruited security experts from a convenience sample of the authors' professional contacts (N=7) and members of NIST's Software and Supply Chain Assurance Forum [75] (N=16). To ensure we received expert opinions, we only considered respondents who self-reported 10 or more years of experience. After each response, we added any suggested tasks for

| # | Secure Development Task | N[1] | B[2] | O[3] | S[4] | F[5] |
|---|---|---|---|---|---|---|
| A1 | *Determine security controls required for the program* | ✓ | | ✓ | ✓ | |
| A2 | *Determine security objectives required for the program* | ✓ | | ✓ | ✓ | |
| A3 | *Perform risk analysis (e.g., probability of successful attack occurrence)* | ✓ | ✓ | ✓ | ✓ | |
| A4 | *Identify potential threats to the program* | ✓ | ✓ | ✓ | ✓ | ✓ |
| A5 | *Identify access points into the system (i.e., attack surface) which could be used by an attacker* | ✓ | ✓ | ✓ | ✓ | ✓ |
| A6 | *Identify the common attack techniques used by potential threats to the application* | ✓ | ✓ | ✓ | ✓ | ✓ |
| A7 | *Identify critical operational requirements which must continue to function in the face of an attack or recover quickly afterwards* | ✓ | ✓ | ✓ | | ✓ |
| A8 | *Identify functions that handle sensitive data (e.g., Personally Identifiable Information)* | ✓ | ✓ | ✓ | ✓ | ✓ |
| A9 | *Identify usage patterns (e.g., misuse, abuse of functionality) that should be disallowed by the software's design* | | | ✓ | | |
| A10 | *Analyze tradeoffs between cost and protection provided by security controls* | ✓ | | | | |
| A11 | *Perform code signing (e.g., integrity checks on software) for use in cryptographically verifying the authenticity of a module or release* | | | ✓ | | |
| A12 | *Identify potential attack vectors associated with the program under development* | ✓ | | ✓ | ✓ | ✓ |
| A13 | *Identify potential attack vectors in environment the program interacts with (e.g., hardware, libraries, etc)* | ✓ | | ✓ | ✓ | ✓ |
| A14 | *Identify potential vulnerabilities in the operationalization of software (e.g., human errors, cultural or political issues)* | ✓ | | ✓ | | ✓ |
| A15 | *Identify security vulnerabilities in others' code* | ✓ | ✓ | | | ✓ |
| A16 | *Identify common coding mistakes that create security vulnerabilities* | ✓ | ✓ | ✓ | | |
| A17 | *Identify potential vulnerabilities as you write code* | ✓ | ✓ | | | |
| A18 | *Use automated code analysis tools to identify vulnerabilities* | | | ✓ | | ✓ |
| A19 | *Use software fuzzing tools to identify vulnerabilities* | | | ✓ | | ✓ |
| A20 | *Identify areas in program design where security risks might be possible* | | | ✓ | | ✓ |
| A21 | *Identify sections of code that might include security vulnerabilities* | | | ✓ | | ✓ |
| A22 | *Understand security issues and concerns associated with reused code (e.g., third-party libraries, shared code)* | | | ✓ | | ✓ |
| A23 | *Apply applicable secure coding and testing standards* | ✓ | ✓ | | ✓ | ✓ |
| A24 | *Apply security principles (e.g., least privilege, layered defense) into design of the program* | | | ✓ | ✓ | ✓ |
| A25 | *Utilize protocols that provide confidentiality* | ✓ | | ✓ | | ✓ |
| A26 | *Utilize protocols that provide integrity* | ✓ | | ✓ | | ✓ |
| A27 | *Utilize protocols that provide availability* | ✓ | | ✓ | | ✓ |
| A28 | *Correctly implement authentication protocols* | ✓ | | ✓ | | ✓ |
| A29 | *Utilize protocols that provide non-repudiation* | ✓ | | ✓ | | |
| A30 | *Leverage enterprise security services to mitigate vulnerabilities (e.g., enterprise PKI)* | ✓ | | ✓ | | |
| A31 | *Leverage enterprise security experts for help to fix vulnerable code* | ✓ | | | | |
| A32 | *Design software to prevent potential vulnerabilities* | ✓ | | ✓ | ✓ | ✓ |
| A33 | *Rewrite software to remove software vulnerabilities* | ✓ | ✓ | | | ✓ |
| A34 | *Design software to quarantine attacker if a vulnerability is exploited* | ✓ | | ✓ | | |
| A35 | *Write code to monitor and log program execution for later review* | ✓ | ✓ | | | ✓ |
| A36 | *Write error handling code to alert for possible malicious behavior* | ✓ | ✓ | | | ✓ |
| A37 | *Design software so that it fails gracefully in the face of attack* | ✓ | ✓ | | | |
| A38 | *Enumerate boundary conditions and mimic potential threats* | ✓ | ✓ | ✓ | ✓ | |
| A39 | *Assess that security requirements are met (e.g., through security design and code reviews)* | ✓ | ✓ | ✓ | | |
| A40 | *Demonstrate the effectiveness of implemented security mitigations* | ✓ | ✓ | ✓ | | ✓ |
| A41 | *Evaluate security controls on interfaces to other software systems the program interfaces with* | ✓ | | ✓ | | |
| A42 | *Evaluate security controls on interfaces to other hardware systems the program interfaces with* | ✓ | | ✓ | | |
| A43 | *Document security implications and assumptions of software so that they are readable and actionable by future developers* | ✓ | | ✓ | | |
| A44 | *Communicate security assumptions and requirements to other developers to avoid vulnerabilities caused by misunderstandings* | ✓ | ✓ | ✓ | | |
| A45 | *Communicate program details with other developers to ensure a thorough security review of the code* | | | ✓ | | |
| A46 | *Communicate with other internal teams the program integrates with to understand how to securely interact with their systems* | ✓ | | ✓ | | |
| A47 | *Discuss lessons learned from internal and external security incidents so all team members are aware of potential threats* | | ✓ | | | |
| A48 | *Document the most important security and privacy error and alert messages requiring user/operator attention* | | | ✓ | | |
| A49 | *Effectively communicate to company leadership the security risks and/or security relevant information assumed when using the program* | ✓ | ✓ | ✓ | | ✓ |
| A50 | *Effectively educate customers on the security risks and/or security relevant information assumed when using the program* | ✓ | ✓ | ✓ | | ✓ |
| A51 | *Effectively communicate to company leadership the cost/risk trade-off associated with deciding whether to fix identified problems* | ✓ | ✓ | ✓ | | |
| A52 | *Effectively communicate to customers the cost/risk trade-off associated with deciding whether patch their software* | ✓ | ✓ | ✓ | | |
| A53 | *Balance user functionality needs with security requirements* | ✓ | | ✓ | | |
| A54 | *Maintain awareness of security issues with new hardware and software technologies and their potential implications* | ✓ | ✓ | | | |
| A55 | *Maintain awareness of attack techniques used by a variety of malicious actors (e.g., insiders, nation states, cyber criminals)* | | ✓ | | | |
| A56 | *Communicate functionality needs to security experts to get recommendations for secure solutions (e.g., secure libraries and languages)* | | | ✓ | | |
| A57 | *Know the appropriate point of contact/response team in my organization to contact if a vulnerability in production code is identified* | | | ✓ | | ✓ |

[1] NICE, [2] BSIMM, [3] OSAMM, [4] SDL, [5] SAFECode

Table 1: Initial secure development tasks identified through framework review. The SAFECode framework was not included in the initial review, but added after multiple expert recommendations.

review by subsequent experts. We stopped recruiting additional experts when no new tasks were suggested and task appropriateness responses remained stable (e.g., no significant difference in results when adding the last 10 responses), the suggested stopping criteria for free-listing exercises [101]. The 22 secure-development experts had 20 years of experience on average, and 81% also held a graduate degree.

We made several changes to our task list based on expert feedback. First, ten tasks were considered inappropriate—*Probably not* or *Definitely not a secure development task*—by 80% of experts. For several tasks, our experts believed it is not the developers' role to determine the balance security and performance costs, but instead the job of customers or leadership (reworded A3 and A7; removed A10, A50, A51, A52, A53). Similarly, our experts stated developers should not be expected to research attacker techniques, but instead get this information from security experts (reworded A5; removed A55). Finally, our experts indicated code signing is not part of secure code production, but instead its deployment, which is out of scope for our construct definition (A11).

Additionally, we revised several tasks' wording. Most significantly, we replaced "program" with "system" to match the modular approach to design common in industry (A1, A2, A4, A12, A13, A20, A29, A35, A45, A46). Other changes included using more common developer terminology (A2), focusing on threats from malicious actors (as opposed to natural disasters) (A6), making security explicit (A4), adding clarifying examples (A15, A29), and rephrasing statements to improve readability (A20, A21, A41, A42).

Finally, we added six tasks. Several experts recommended tasks for identifying and using secure programming languages and libraries (B15, B16, B26, B27, B28; shown in Table 2). One expert also suggested adding a task for correctly implementing authorization protocols (B34), as it represents a distinct access-control component (compared to authentication).

Multiple experts suggested the Software Assurance Forum for Excellence in Code (SAFECode) Fundamental Practices for Secure Software Development framework [83] as an additional task source, so we repeated the framework review process for SAFECode. While no task list or task categories changes were made, it provided further support for tasks already included.

**Developer pilot.** Next, to ensure our target population could easily understand each item, we piloted the post-expert-review tasks with eight developers. First, we reframed each task as an *"I can"* statement regarding the developer's confidence in performing the task. We then asked them to indicate their confidence using a 5-point Likert-scale from *"I am not confident at all"* to *"I am absolutely confident."* We also provided a *"Do not understand"* option if the respondent did not understand the task's meaning. Our full survey text is given in our supplementary material's main survey section.

We recruited a convenience sample of the researchers' professional contacts, chosen to represent varying experience levels. Participants were asked to "think aloud" as they responded to each question. We updated the questions after each pilot, eventually reaching the final set given in Table 2. Specifically, we made the following changes: we reworded A48 to make it clear we were asking about writing understandable security-error messages, updated A8 to indicate we were asking about code the developer has themself written as opposed to a library function, and replaced the term "boundary cases" with "edge cases" in A38 to use the more common terminology.

## REFINING THE SCALE

To trim our item set and determine the underlying factor structure, we recruited 157 developers and performed EFA. This section describes the methods used and our analysis results.

### Recruitment

From September 2018 to July 2019, we recruited participants using several methods to broadly sample the developer population. First, we contacted software-development-related groups' leadership. This included popular Meetup.com [67] and LinkedIn [59] groups, regional ACM chapters [35], and the researchers' personal contacts at large development companies. We asked each contact to share study details with their organization's members and their colleagues. Prior work has found relative success partnering with organizational leadership in this manner, adding credibility to recruitment messages [99]. We also posted messages on relevant online forums such as Reddit and Slack channels. Dietrich et al. showed this method's usefulness with technology professionals, as participants are reached in a more natural setting and are more likely to be receptive [28]. Finally, we recruited developers directly through the freelancing platform Upwork [97] and the research-participant recruitment site Prolific [77]. Because of our broad recruitment process, we do not include respondents who reported less than one year of development experience.

We also varied the study's compensation method. Prior work suggests using a mix of incentives increases participant diversity [47]. Participants recruited through organizational contacts and online forums were recruited in two waves. In the first wave, we did not advertise or provide any compensation for participation. In the second, participants were entered into a lottery for one of 10 $20 Amazon gift cards. Participants recruited through Upwork and Prolific were paid $8 each.

### Survey design

Participants were shown the 58 "I can" statements in Table 2 and asked to rate each on a 5-point Likert-scale from *"I am not confident at all"* to *"I am absolutely confident"* or indicate they *"Do not understand"*. Tasks were presented in random order to prevent ordering effects. At the survey's conclusion, participants were asked to indicate their software development skill level on a 5-point Likert scale ranging from "Novice (limited experience)" to "Expert (recognized authority)," their years of software development experience, and their average time spent daily performing software development tasks, along with other demographic questions. Our full survey text is given in our supplementary materials' main survey section.

We were concerned some participants might overrate their secure development skill to portray appear more socially acceptable [27]. To avoid social desirability bias, our study utilized

| # | Secure Development Statement | Do not und. | $\mu$ | $\sigma$ |
|---|---|---|---|---|
| **F1** | | | | |
| B3 | *I can perform a threat risk analysis (e.g., likelihood of vulnerability, impact of exploitation, etc.)* | 0% | 3.15 | 1.29 |
| B4 | *I can identify potential security threats to the system* | 0% | 3.61 | 1.10 |
| B6 | *I can identify the common attack techniques used by attackers* | 0% | 3.45 | 1.15 |
| B11 | *I can identify potential attack vectors in the environment the system interacts with (e.g., hardware, libraries, etc.)* | 1.72% | 2.97 | 1.25 |
| B15 | *I can identify common vulnerabilities of a programming language* | 0% | 3.44 | 1.15 |
| B39 | *I can design software to quarantine an attacker if a vulnerability is exploited* | 1.72% | 2.65 | 1.34 |
| B44 | *I can mimic potential threats to the system* | 1.72% | 3.19 | 1.19 |
| B47 | *I can evaluate security controls on the system's interfaces/interactions with other software systems* | 1.72% | 3.28 | 1.23 |
| B48 | *I can evaluate security controls on the system's interfaces/interactions with hardware systems* | 3.45% | 2.91 | 1.30 |
| **F2** | | | | |
| B8 | *I can identify code that handles sensitive data (e.g., Personally Identifiable Information)* | 0% | 4.09 | 1.10 |
| B31 | *I can correctly implement authentication protocols* | 0% | 3.76 | 1.16 |
| B32 | *I can correctly implement authorization protocols* | 3.45% | 3.78 | 1.11 |
| B50 | *I can communicate security assumptions and requirements to other developers on the team to ensure vulnerabilities are not introduced due to misunderstandings* | 0% | 3.71 | 1.10 |
| B51 | *I can communicate system details with other developers to ensure a thorough security review of the code* | 0% | 3.85 | 1.16 |
| B53 | *I can discuss lessons learned from internal and external security incidents to ensure all development team members are aware of potential threats* | 0% | 3.88 | 1.11 |
| B55 | *I can effectively communicate to company leadership identified security issues and the cost/risk trade-off associated with deciding whether or not to fix the problem* | 0% | 3.78 | 1.13 |
| B57 | *I can communicate functionality needs to security experts to get recommendations for secure solutions (e.g., secure libraries, languages, design patterns, and platforms)* | 3.45% | 3.81 | 1.11 |
| B58 | *I know the appropriate point of contact/response team in my organization to contact if a vulnerability in production code is identified* | 1.72% | 4.04 | 1.25 |
| B1 | *I can determine security controls which are necessary to implement in the system* | 6.90% | 3.50 | 1.13 |
| B2 | *I can determine security requirements for the system* | 1.72% | 3.64 | 1.06 |
| B5 | *I can identify access points into the system (i.e., attack surface) which could be used by an attacker* | 1.72% | 3.34 | 1.24 |
| B7 | *I can identify critical operational requirements which must continue to function or recover quickly after an attack* | 6.90% | 3.44 | 1.15 |
| B9 | *I can identify usage patterns that should be disallowed by the system's design* | 0% | 3.66 | 1.12 |
| B10 | *I can identify potential attack vectors associated with the system under development* | 6.90% | 3.24 | 1.26 |
| B12 | *I can identify potential vulnerabilities in the operationalization of software (e.g., human errors)* | 5.17% | 3.77 | 1.10 |
| B13 | *I can identify security vulnerabilities in others? code (e.g., peer review or third party components)* | 0% | 3.38 | 1.20 |
| B14 | *I can identify common coding mistakes that create security vulnerabilities* | 0% | 3.80 | 1.07 |
| B16 | *I can understand security limitations of a programming language* | 0% | 3.68 | 1.04 |
| B17 | *I can identify potential vulnerabilities as I write code* | 0% | 3.73 | 1.01 |
| B18 | *I can use automated code analysis tools to identify vulnerabilities* | 1.72% | 3.31 | 1.27 |
| B19 | *I can use software fuzzing tools to identify vulnerabilities* | 13.79% | 2.92 | 1.33 |
| B20 | *I can review system design to identify areas where potential security risks exist* | 0% | 3.44 | 1.19 |
| B21 | *I can identify sections of code that are most likely to include security vulnerabilities* | 0% | 3.54 | 1.14 |
| B22 | *I can understand security issues and concerns associated with reused code (e.g., code samples, shared code)* | 1.72% | 3.79 | 1.07 |
| B23 | *I can apply applicable secure coding and testing standards* | 3.45% | 3.63 | 1.19 |
| B24 | *I can use provably secure programming languages* | 22.41% | 3.72 | 1.26 |
| B25 | *I can identify secure implementations of common libraries* | 3.45% | 3.34 | 1.19 |
| B26 | *I can use a secure implementation of a common library that is recommended by a security expert* | 6.90% | 4.01 | 1.14 |
| B27 | *I can apply security principles (e.g., least privilege) into the design of the system* | 3.45% | 3.49 | 1.25 |
| B28 | *I can utilize protocols that provide confidentiality of user data* | 5.17% | 3.83 | 1.18 |
| B29 | *I can utilize protocols that provide integrity of user data* | 3.45% | 3.81 | 1.13 |
| B30 | *I can utilize protocols that provide availability in the face of an attack* | 10.34% | 3.07 | 1.27 |
| B33 | *I can utilize protocols that provide non-repudiation* | 39.66% | 3.21 | 1.31 |
| B34 | *I can leverage enterprise security services to mitigate vulnerabilities (e.g., enterprise PKI)* | 12.06% | 3.16 | 1.32 |
| B35 | *I can leverage enterprise security teams for help to fix vulnerable code* | 3.45% | 3.93 | 1.23 |
| B36 | *I can leverage external security review (e.g., penetration testing, bug bounties) to find vulnerable code* | 3.45% | 3.41 | 1.26 |
| B37 | *I can design software to prevent potential vulnerabilities* | 1.72% | 3.41 | 1.24 |
| B38 | *I can rewrite software to remove vulnerabilities* | 0% | 3.66 | 1.18 |
| B40 | *I can write code to monitor and log system execution for later review* | 1.72% | 3.95 | 1.23 |
| B41 | *I can write error handling code to alert for possible malicious behavior* | 0% | 3.77 | 1.13 |
| B42 | *I can design software so that it fails gracefully in the face of attack* | 3.45% | 3.19 | 1.30 |
| B43 | *I can enumerate edge cases of the system's use* | 13.79% | 3.43 | 1.15 |
| B45 | *I can assess that security requirements are met (e.g., through security design and code reviews)* | 0% | 3.65 | 1.13 |
| B46 | *I can demonstrate the effectiveness of implemented security mitigations* | 12.07% | 3.35 | 1.20 |
| B49 | *I can document a system's security implications and assumptions so they are readable and actionable by others* | 1.72% | 3.68 | 1.18 |
| B52 | *I can communicate with other internal teams to understand how to securely interact with their systems* | 0% | 3.95 | 1.10 |
| B54 | *I can write understandable security and privacy error messages to draw the required user/operator attention* | 0% | 3.92 | 1.13 |
| B56 | *I can maintain awareness of hardware and software technologies' security issues and their potential implications* | 0% | 3.40 | 1.30 |

Table 2: Set of secure development tasks identified after both the expert review and developer pilot transformed into *"I can"* statements. Each task was evaluated on a 5-point Likert-scale (from "I am not at all confident" to "I am absolutely confident") by 157 developers. For each statement, we give the rate of "Do not understand" responses, the average response, and standard deviations. The final items retained based on EFA are shown first grouped according to their associated sub-scale.

deception [72]. That is, during recruitment and throughout the survey, participants were told our goal was to measure *general* software development self-efficacy. At the survey's conclusion, participants were told the study's true nature and were allowed withdraw and have their response deleted.

## Demographics

We received 181 responses, but 7 (4%) did not have one year of development experience and 9 withdrew (5%) after learning the study's true purpose. We removed eight responses (4%) considered careless based on abnormally short response times [66]. We set the cutoff at less than five minutes based on an obvious threshold in the data. The final 157 participants' development experience ranged from 1 to 45 years ($\mu = 8.68$, $\sigma = 9.46$). Our participants were predominantly male (88%), young (50% below 30 and 83% below 40), educated (77% held a bachelor's degree, 26% held a graduate degree), and white (51%) or Asian (29%). Our participants' experience and demographic characteristics are similar to those found by prior large-scale developer surveys [92, 53].

Choosing an appropriate sample size can be complicated. Prior work suggests basing sample size on the number of items [5, 13, 17, 38, 42, 55, 58, 63, 74] with a minimum of 100 to 200 participants required [25, 26, 38, 41, 42, 58, 61]. For example, Hair et al. recommend 5 participants per item [42]. However, empirical evaluations of scales' component analysis stability (i.e., whether the factor structure identified with EFA varied) with various sample sizes has not found evidence to support the sample size-to-item ratio [8, 3, 98]. Instead, prior work shows factor loading (i.e., the magnitude of correlation between items and their associated factor), the absolute sample size, and the number of variables associated with each factor have the most significant effect on component analysis stability [40]. Therefore, Guadagnoli and Velicer recommend 150 participants as sufficient if factors are made up of high numbers of items (10 or 12), even with low loadings ($l < 0.40$) [40]. We believed that, because of the large number of items tested, we would be likely to meet this standard, so we targeted about 150 qualifying participants. In fact (as will be discussed below), our data met Guadagnoli's and Velicer's more conservative standard: four or more variables per component having loadings over 0.60, which they found sufficient to assess underlying component structures "whatever the sample size used." We therefore conclude our sample size was sufficient.

## Poorly behaving items

To refine our scale, we first checked for poorly behaving items across several metrics. First, we observed our scale's internal consistency was very high (Cronbach's $\alpha = 0.98$), indicating our items were closely related. Next, we checked each items' item-total correlation, which indicates how *discriminant* the item is (i.e., participants who score high on the item are more likely to score high on the full scale). Items with low item-total correlation ($< 0.2$) should be excluded because they do not adequately reflect the scale [32]. We did not observe any questions with low item-total correlation.

We next looked for ceiling or floor effects: tight response groupings at either extremes of the Likert scale (e.g., $4.0 < \mu < 5.0$ and $\sigma < 1$). Since scales are designed to measure differences between participants, individual questions need to exhibit adequate variance; if everyone responds similarly, the item has limited utility. We found no items with this effect.

Finally, it is important to ensure our target population understands each item. If a respondent if confused by the terminology used or question phrasing, their response is framed by a misconception and not reflective of the underlying construct. We removed 13 items to which $> 5\%$ of participants responded "Do not understand" or simply skipped. In many cases (B12, B19, B28, B30, and B33), the confusion seemed to stem from using less common security terminology such as *fuzzing* or *non-repudiation*. Because the scale should usable by developers of all levels of security knowledge, we removed these items. Similarly, 12.06% of participants found B34 confusing, likely because it asks whether participants can use security services provided by their enterprise. As not all developers work in an enterprise setting which offers these services—including many of the freelancers we recruited—this item also does not meet our goal of producing a measure for all developers.

## Factor analysis

With the remaining 45 well behaving items, we set out to identify the scale's underlying factor structure. Here, we define factors as our construct's sub-components. A construct can have one component, indicating the scale's items measure it directly, or multiple components, where the scale can be broken into component sub-scales and together their scores reflect the construct. Because these factors are latent, they can not, by definition, be measured directly. Instead, we must first determine our construct's number of factors and which items best describe each factor, i.e., the underlying factor structure.

Prior to attempting to identify our factor structure, we checked whether our data actually measured common factors and were correlated (prerequisites to establishing the number of factors and their structure). According to Bartlett's test of sphericity ($\chi^2 = 4833.35, p < 0.001$) [91] and the Measure of Sampling Adequacy (0.936) [19], we confirmed our data met these goals.

Next, to identify the factor structure, we performed an exploratory Principal Component Analysis (PCA). PCA is a data summarization method that transforms item responses such that the first dimension (or *component*) explains as much variance in the original data as possible, with each subsequent and orthogonal component explaining as much of the remaining variance as possible. In EFA, these components represent the scale's underlying factors. To produce an efficient factor structure (i.e., one identifying the most variation with the least set of items), we only retain the top components.

Since there is no standard method for deciding the number of retained components, we relied on several and followed the most common recommendation. This consensus protocol accounts for each method's strengths and weaknesses. First, we calculated each components' eigenvalues and selected those with eigenvalues $> 1.0$ according to the Kaiser criterion [38]. Next, we determined optimal coordinates by fitting a line to the smallest eigenvalues with a linear regression and identifying where our eigenvalues diverge [80]. We also performed a parallel analysis by generating random data, calculating the associated eigenvalues, and retaining any eigenvalues whose value was greater than the random data's eigenvalues [46]. Finally, we determined the acceleration factor by looking for the point where our eigenvalues changed dramatically [80]. In these analyses, the Kaiser criterion recommended six components, the optimal coordinates and parallel analysis both recommended two, and the acceleration factor recommended one. Therefore, we retained two components.

To determine which factors each item associated most with (i.e., which it loads on), we rotated responses [39]. There are multiple possible rotation types, which can be divided into orthogonal (e.g., varimax) or oblique (e.g., direct oblimin). Orthogonal rotations are appropriate when the factors are not expected to be correlated and an oblique rotation is appropriate otherwise [39]. Because we did not know whether our factors were correlated, we followed the recommendation of Tabachnick and Fiddell who suggest first using an oblique rotation (in our case a direct oblimin), calculating the correlation of the identified sub-scales associated with each factor, and switching to an orthogonal rotation if correlations do not exceed 0.32, indicating 10% (or more) overlap in variance among factors [93]. We found our factors were correlated (0.64) and maintained the direct oblimin rotation.

After rotating items, we selected which ones to retain, using three inclusion criteria. First, we only considered an item as loading on a factor if its loading exceeded 0.5, indicating significant association with the underlying factor [72]. Next, we applied Saucier's criterion, only considering an item to load on a factor if its loading exceeded twice the loading on any other factor. This ensures variance in item responses maps to changes in the associated factor. Finally, we chose to remove items where the item variance accounted for by all the retained factors (its communality) was less than 0.4, as recommended by Fabrigar et al. [33], as this tends to indicate low item reliability. This led us to remove 27 more items. We reran PCA on the remaining items and found that the two retained components predicted more than 56.9% of variance. Notably, the first factor accounts for a majority of the scale's variance (47.6%), with the second factor accounting for 9.3% of variance. The rotated factor loadings are given in Table 3. Note, because all our factor loadings are above 0.60, this confirms the sufficiency of our 157 developer sample [40].

### Reliability

To confirm our remaining items maintained their internal reliability, we first computed Cronbach's $\alpha$ for the full scale ($\alpha$ = 0.936) and each sub-scale ($\alpha$ = 0.907, 0.876). We found that our data met McKinley et al.'s suggested threshold that a multi-component scale is reliable if $\alpha$ exceeds 0.6 and a majority of sub-scale $\alpha s$ exceed 0.7 [65].

Next, we tested item-total correlation using Pearson correlation between each item and the average of all other items in the same sub-scale. All items exceeded Everritt's 0.2 threshold [32]. Finally, we observed each sub-scales' mean item-total correlation (0.520 and 0.440, respectively) exceeded 0.30, which is considered "exemplary" [84]. Based on these measures, we confirmed our reduced scale had high reliability.

|  | F1 | ITC |  | F2 | ITC |
|---|---|---|---|---|---|
| $\alpha$ | 0.907 | – | $\alpha$ | 0.876 | – |
| IIC | 0.520 | – | IIC | 0.440 | – |
| B3 | 0.81 | 0.78 | B8 | 0.64 | 0.66 |
| B4 | 0.69 | 0.80 | B31 | 0.61 | 0.70 |
| B6 | 0.83 | 0.78 | B32 | 0.67 | 0.67 |
| B11 | 0.79 | 0.81 | B50 | 0.73 | 0.79 |
| B15 | 0.65 | 0.66 | B51 | 0.68 | 0.74 |
| B39 | 0.74 | 0.73 | B53 | 0.76 | 0.73 |
| B44 | 0.64 | 0.72 | B55 | 0.64 | 0.72 |
| B47 | 0.64 | 0.77 | B57 | 0.61 | 0.70 |
| B48 | 0.82 | 0.78 | B58 | 0.80 | 0.67 |

Table 3: Remaining items and factor structure after initial EFA. The first two rows show reliability measures (Cronbach's $\alpha$ and average inter-item correlation) for each sub-scale. The remaining rows show the retained items, their loadings, and item-total correlations within the sub-scale.

## FINALIZING THE SCALE

We next conducted an additional round of surveys from July to September 2019 with the 18 items remaining after EFA (shown at the top of Table 2). In this step, we tested whether the identified two-factor structure was maintained, remaining reliable, with a different participant pool.

### Survey Design

Respondents were again asked to respond to the 18 items on a 5-point Likert scale from "I am not at all confident" to "I am absolutely confident." We removed the "Do not understand" option, as we had sufficiently established item face validity.

It is important to confirm our scale measures the targeted construct by testing whether responses match other theoretically relevant measures. To test whether SSD-SES converges with measures we expect it to relate with (*convergent validity*), while being distinct from other, similar scales (*discriminant validity*), we performed Pearson's correlation tests with four related scales. We similarly use Pearson's correlation to compare to two additional, well-established psychometric scales to understand how participant psychological characteristics relate to secure-development self-efficacy. All p-values reported in this section are corrected using a Bonferroni-Holm correction to account for multiple testing [45]. To avoid overburdening participants, we randomly present two of the six additional scales (described in detail below) to each participant.

### Recruitment

We used the same recruitment process as the prior step, targeting 120 responses. Prior work suggests 100 participants are sufficient for confirmatory factor analysis [12]. We targeted 20 more participants to have 40 participants complete each additional psychometric scale, giving sufficient power ($> 0.80$) to identify $> 42.5\%$ correlation with SSD-SES [24].

Due to the relative difficulty of recruiting through organizations and online forums, we predominantly relied on Upwork's freelancing service and Prolific in this step. As developers on these platforms tend to be less experienced, respondents from these sites first completed a pre-screening survey, asking them to report years of software development experience and average time spent daily on development tasks. We then invited screened respondents to the full survey using a quota-sampling method. Our quotas were chosen to match developer experience ranges from StackOverflow's most recent developer demographics survey [92]. Specifically, we sought to survey 14 (11.4%) inexperienced developers (1-2 years), 70 (58.4%) moderately experienced developers (3-11 years), and 36 (30.2%) experienced developers ($>$ 11 years). We concluded recruitment after reaching each quota, though we were not able to maintain the desired percentages due to the unpredictable nature of responses from organizational contacts.

### Demographics

A total of 162 developers responded in this round, but 2 participants (1%) had less than a year of software-development experience, 6 chose to withdraw (4%), and 8 (5%) responses were considered careless (i.e., completed in less than 5 minutes [66]). The remaining 146 participants' development experience was split between our three quota ranges as follows: 10.3% inexperienced, 65.1% moderately experienced, and 24.6% experienced ($\mu = 9.79$, $\sigma = 10.80$). Again, our participants were predominantly male (91%), young (48% below 30 and 77% below 40), educated (57% held a bachelor's degree, 26% held a graduate degree), and white (77%) or Asian (9%), matching broader developer demographics [92, 53].

### Factor Analysis

To determine whether the latent factor structure identified previously held with our new population, we repeated the PCA procedure using a direct oblimin rotation. We observed three items either no longer sufficiently loaded on their original factor (i.e., B8's loading on F2 reduced to 0.39) or switched factors (i.e., B31 and B32 switched to load on F1). Because these items did not behave reliably across multiple samples, they were removed [72]. The remaining items demonstrated internal consistency with a Cronbach's $\alpha$ of 0.92 (sub-scale $\alpha$s were 0.90 and 0.88, respectively). The 15 items and their associated latent factors are given in Table 4 with their mean responses, factor loadings, and item-total correlations within their sub-scales. All the remaining items and their sub-scales behaved appropriately according to the variability and reliability metrics given in the prior section.

The remaining sub-scale items represented two distinct themes: vulnerability identification and mitigation tasks and security communication tasks. Again, the first factor accounts for a majority of the scale's variance (48.1%), and the security communication sub-scale accounts for 11.1% of variance.

While EFA is useful for determining a possible underlying factor structure, it is not able to assess that structure's goodness-of-fit onto the data with respect to other possible structures [60]. Therefore, we also performed a Confirmatory Factor Analysis (CFA), which confirms our prior analyses have been conducted thoroughly and appropriately [44]. CFA is a type of structural-equations analysis assessing rival models' goodness-of-fit. Specifically, we compare a null model with all items loading on separate factors, a single common factor model, and our multi-factor model [52].

Our model demonstrated sufficient goodness-of-fit with its $\chi^2$ (176.38) below the conservative limit of twice the its degrees of freedom (DoF = 89) [16]. Using ANOVA comparisons, we found our model fit better than the null ($\chi^2 = 1257.34$, $p < 0.001$) and the single-factor model ($\chi^2 = 317.78$, $p < 0.001$).

We also calculated several other goodness-of-fit metrics. First, we determined the Comparative Fit Index (CFI), which measures the model's fit relative to a more restrictive baseline model [9], and the Tucker-Lewis Index (TFI), a more conservative version of CFI, penalizing overly complex models [10]. Our model performed well in both (CFI = 0.92, TFI = 0.91), with scores over the recommended threshold of 0.90 [72]. Next, we calculated the Standardized Root Mean Square Residual (SRMR), an absolute measure of fit calculating the difference between observed and predicted correlation [96]. Our model demonstrated a sufficient SRMR of 0.054, as a value below 0.08 is considered good fit [96]. Finally, we calculated the Root Mean Square Error of Approximation (RMSEA), which measures how well the model reproduces covariances among items, instead of comparing to a baseline model. Our model demonstrated a "moderate" fit with a RMSEA of 0.082 [62].

### Reliability

To measure the sub-scales' internal consistency, we calculate their composite reliability [36]. Composite reliability offers a more accurate view of reliability over Cronbach's $\alpha$, which makes several potentially inaccurate assumptions, such as considering factor loadings and error variances equal [82]. Instead, composite reliability considers factor loadings from CFA, measuring the latent factors' reliability instead of their individual items. We found both sub-scales' composite reliability (0.90 and 0.87, respectively) exceeded the recommended threshold of 0.60, indicating they are internally consistent [6].

### Convergent Validity

Prior work suggests the ability to identify vulnerabilities is influenced by participants' understanding of the development environment (e.g., the programming language and libraries used) and level of security experience [99]. We next consider how well our scale correlates with each of these concepts.

**Secure-development self-efficacy is related to general development self-efficacy.** To measure respondents' software development skill, we utilized the Computer Programing Self-Efficacy Scale (CPSES), a measure of respondents' belief in their ability to produce working programs meeting functionality requirements [95]. CPSES scores were statistically significantly correlated with both SSD-SES sub-scales ($\rho = 0.528$, $p = .001$ and $\rho = 0.627$, $p < 0.001$, respectively).

To measure security experience, we asked if participants had received security training, if they had found a vulnerability or had one found in their code, and how often they communicate with security experts. We estimated the relationship of security experience and secure-development self-efficacy with a poisson regression (appropriate for count data [15]). For each sub-scale, our initial regression model included each security experience response and the participants' software development experience as independent variables. To avoid

| # | Secure Development Statement | $\mu$ | $\sigma$ | $l$ | ITC |
|---|---|---|---|---|---|
| **Vulnerability Identification and Mitigation** (48.2% of variance explained, CR = 0.90) | | | | | |
| C3 | *I can perform a threat risk analysis (e.g., likelihood of vulnerability, impact of exploitation, etc.)* | 2.99 | 1.15 | 0.74 | 0.79 |
| C4 | *I can identify potential security threats to the system* | 3.34 | 0.99 | 0.76 | 0.79 |
| C6 | *I can identify the common attack techniques used by attackers* | 3.30 | 1.15 | 0.66 | 0.77 |
| C11 | *I can identify potential attack vectors in the environment the system interacts with (e.g., hardware, libraries, etc.)* | 2.91 | 1.10 | 0.67 | 0.75 |
| C15 | *I can identify common vulnerabilities of a programming language* | 3.34 | 1.16 | 0.56 | 0.72 |
| C39 | *I can design software to quarantine an attacker if a vulnerability is exploited* | 2.41 | 1.19 | 0.77 | 0.72 |
| C44 | *I can mimic potential threats to the system* | 2.97 | 1.06 | 0.78 | 0.71 |
| C47 | *I can evaluate security controls on the system's interfaces/interactions with other software systems* | 3.08 | 1.02 | 0.72 | 0.72 |
| C48 | *I can evaluate security controls on the system's interfaces/interactions with hardware systems* | 2.88 | 1.17 | 0.78 | 0.73 |
| **Security Communication** (11.1% of variance explained, CR = 0.87) | | | | | |
| C50 | *I can communicate security assumptions and requirements to other developers on the team to ensure vulnerabilities are not introduced due to misunderstandings* | 3.46 | 1.03 | 0.73 | 0.86 |
| C51 | *I can communicate system details with other developers to ensure a thorough security review of the code* | 3.50 | 1.15 | 0.78 | 0.83 |
| C53 | *I can discuss lessons learned from internal and external security incidents to ensure all development team members are aware of potential threats* | 3.64 | 1.11 | 0.65 | 0.72 |
| C55 | *I can effectively communicate to company leadership identified security issues and the cost/risk trade-off associated with deciding whether or not to fix the problem* | 3.51 | 1.18 | 0.60 | 0.75 |
| C57 | *I can communicate functionality needs to security experts to get recommendations for secure solutions (e.g., secure libraries, languages, design patterns, and platforms)* | 3.60 | 1.12 | 0.73 | 0.83 |
| C58 | *I know the appropriate point of contact/response team in my organization to contact if a vulnerability in production code is identified* | 3.90 | 1.14 | 0.91 | 0.73 |

Table 4: SSD-SES's final questions and associated sub-scales. Responses were reported on the following scale: *I am not confident at all* (1), *I am slightly confident* (2), *I am somewhat confident* (3), *I am moderately confident* (4), and *I am absolutely confident* (5).

overfitting, we tested all combinations of the independent variables, selecting the model with minimum Bayesian Information Criteria [79], as is standard.

Table 5a shows the vulnerability identification and mitigation regression model and Table 5b gives the security communication regression model. Base cases were selected to represent the medium experience level to allow clearer comparisons. Variable levels are presented in decreasing experience order. The log estimate (LE) column gives the variable's observed effect. For categorical variables, the LE is the expected relative change in SSD-SES sub-scale score when moving to the given variable level from the base case. The LE for each base case is definitionally 1.0. We also give the 95% confidence interval for the log estimate (CI) and the associated $p$-value.

**Secure-development self-efficacy increases with security experience.** As expected, both SSD-SES sub-scale scores increase with security experience. Specifically, participants who had never found a vulnerability (LE = 0.82, $p < 0.001$) or never worked with a security expert (OR = 0.81, $p < 0.001$) had less belief in their ability to find and mitigate security vulnerabilities. Further, those who have found multiple vulnerabilities (LE = 1.08, p = 0.49) or worked with multiple security experts (LE = 1.09, p = 0.024) had even higher self-efficacy.

We observed nearly the same significant trends in the security communication sub-scale. However, more than one experience finding a vulnerability did not show a significant increase over a single vulnerability identification experience (LE = 1.08, p = 0.112). Additionally, we found experienced developers ($>$ 11 years experience) were more likely to believe they could effectively discuss security (LE = 1.11, p = 0.017).

## Discriminant Validity

To confirm that SSD-SES in fact measures a new construct, we compared it to two end-user security behavior scales and a general self-efficacy scale. First, we tested whether either sub-scale correlated with the Security Intention Behavior Scale (SeBIS) [31], which measures end-user intention to perform a variety of security behaviors, and the End-User Security Attitudes (SA-6) scale [34], which measures end-user attitudes toward common security behaviors. We did not observe any significant correlation between either scale and SSD-SES.

Next, we tested the correlation between SSD-SES and the General Self-Efficacy Scale (GSE), which assesses the "belief that one can perform a novel or difficult tasks, or cope with adversity—in various domains of human functioning [88]." This comparison tested whether SSD-SES simply measured respondents' belief in themselves as opposed to a domain-specific belief. Again, we did not observe any significant correlation between GSE and SSD-SES.

The lack of significant correlations with SeBIS, SA-6, or GSE indicates SSD-SES measures a distinct underlying construct.

## Relationship with Psychological Constructs

Finally, we included two well-established psychometric measures to understand how participants psychological characteristic relate to SSD-SES scores: the Need for Cognition (NFC) scale and the General Decision-Making Scale (GDMS).

**Curious developers believed more in their ability to identify and mitigate vulnerabilities.** NFC measures intellectual curiosity (i.e., the tendency to engage in and enjoy thinking) [14]. We found that NFC significantly correlated with the vulnerability identification and mitigation sub-scale ($\rho = 0.464$, $p = 0.007$), but not the security communication

| Variable | Value | Log Estimate | CI | $p$-value |
|---|---|---|---|---|
| Found Vuln | **Multiple** | **1.08** | **[1, 1.17]** | **0.049** |
| | Once | – | – | – |
| | **Never** | **0.82** | **[0.73, 0.92]** | **< 0.001*** |
| Expert Coworker | **Multiple** | **1.09** | **[1.01, 1.17]** | **0.024*** |
| | One | – | – | – |
| | **Never** | **0.81** | **[0.74, 0.88]** | **< 0.001*** |

*Significant effect – Base case (Log Estimate = 1, by definition)

(a) Vulnerability Identification and Mitigation

| Variable | Value | Log Estimate | CI | $p$-value |
|---|---|---|---|---|
| Found Vuln | Multiple | 1.08 | [0.98, 1.18] | 0.112 |
| | Once | – | – | – |
| | **Never** | **0.86** | **[0.76, 0.98]** | **0.026*** |
| Expert Coworker | **Multiple** | **1.06** | **[0.97, 1.15]** | **0.191*** |
| | One | – | – | – |
| | **Never** | **0.87** | **[0.79, 0.95]** | **0.002*** |
| Dev Experience | **> 11 years** | **1.11** | **[1.02, 1.2]** | **0.017*** |
| | 3-11 years | – | – | – |
| | 0-2 years | 0.91 | [0.79, 1.03] | 0.143 |

*Significant effect – Base case (Log Estimate = 1, by definition)

(b) Security Communication

Table 5: Summary of regressions estimating relationship between each sub-scale and security experiences.

sub-scale ($\rho = 0.183$, $p = 0.337$). This suggests developers who are more curious are more likely to feel confident in their ability to search for, identify, and mitigate vulnerabilities. This finding corroborates prior work, which observed developers who were more open or curious were more likely to find vulnerabilities in secure-development puzzles [76].

We also compared SSD-SES to GDMS, which assesses how individuals approach decisions with respect to five decision styles. As secure development requires complex planning and decision-making, our goal was to test whether any style correlated with better secure-development self-efficacy. However, we did not observe any statistically significant correlation.

## DISCUSSION AND LIMITATIONS

Our final 15-item scale measures two distinct underlying factors: vulnerability identification and mitigation as well as security communication. Through our scale development process we observed SSD-SES demonstrate construct validity, internal consistency and reliability, goodness-of-fit, and convergent and discriminant validity. We found SSD-SES correlated with general programming self-efficacy, security experience, and intellectual curiosity, as expected. In this section, we discuss our study's limitations and the need for future work as well as how SSD-SES can be employed by researchers and practitioners.

### Limitations

Software developer recruitment is challenging [69], requiring significant effort to reach the sample used in this study. While we sought to recruit a diverse sample with respect to development experience and we observed similar demographics to the global developer population, it is possible our recruiting methods do not fully represent the broader population. Future work should consider methods to recruit participants not active in development organizations, online forums, or freelancing and research recruitment platforms.

In the final round of recruitment, we performed quota sampling based on participants' years of experience. We also randomly assigned two of six external scales to each participant, without consideration for quota sampling. As a result, the experience distributions for these external scales were not entirely consistent: fewer than expected participants in the inexperienced group were assigned CPSES (2% as opposed to 10% of the total sample) and fewer of the experienced group were assigned GES (14% as opposed to 25%). The supplementary materials give further details. We do not expect this fairly small inconsistency to have a large effect on the results, particularly as we observed almost no significant effects related to experience

Though we found SSD-SES reliable and valid according to several measures, additional testing is necessary. First, as we have only shown correlation between factors and other psychometric measures, we cannot make assessments of causal relationships; as such, we cannot yet create a predictive model to target interventions at specific self-efficacy components.

Most importantly, further work is necessary to determine if SSD-SES measures *actual* secure development skill improvement. In future work, we plan to administer SSD-SES as a pre- and post-test for a hands-on security training course, allowing us to assess improvements and correlate with course grades.

Finally, technology (and accordingly, secure development practice) changes over time. We designed SSD-SES to describe general principles we believe should remain relatively static, but as with any scale SSD-SES should be occasionally revisited and refreshed as needed.

### Using SSD-SES

Our creation and validation of a lightweight secure software development self-efficacy scale presents a valuable resource for a variety of purposes. Below, we suggest possible uses.

**SSD-SES for testing educational interventions.** The initial impetus for creating SSD-SES was o measure the value of an educational intervention, such as a capture-the-flag exercise [22]. By applying SSD-SES before and after administering training, a researcher can observe changes in secure-development self-efficacy, providing feedback for the improvement and comparison of interventions.

To emphasize this benefit, we note that we did not observe a significant relationship between SSD-SES and participants' self-reported security training in our regression. While we would expect training to improve self-efficacy, our results suggest in practice, it does not consistently do so. Therefore, future work in security education is necessary to identify the right training to produce improved outcomes.

**SSD-SES as a covariate.** SSD-SES also provides a useful option for software-developer studies where the researcher may want to control for participants' secure-development skill. For example, in a usability study of security APIs, it would be beneficial to include SSD-SES as a covariate to ensure differences in outcomes between participants occur because of changes to the API and not differences in security skill.

**SSD-SES for measuring security culture.** Finally, because SSD-SES is a lightweight measure, it can be administered broadly to capture an organization's "security culture". This would be useful both for researchers comparing different organizations or measuring change over time, but also for practitioners looking to diagnose whether and what actions are necessary for organizational improvement.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers who provided helpful comments on drafts of this paper, the security experts who helped us define the domain, and all the organizations that contributed to recruitment for this study. This project was supported by NSF grant CNS-1801545.

## REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson L. Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP '17)*. IEEE Computer Society, 154–171. http://dblp.uni-trier.de/db/conf/sp/sp2017.html#AcarOFGKMS17

[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 *Proceedings of the 37th IEEE Symposium on Security and Privacy (SP '16)*. IEEE Computer Society, Los Alamitos, CA, USA, 289–305. DOI: http://dx.doi.org/10.1109/SP.2016.25

[3] Willem A. Arrindell and Jan van der Ende. 1985. An Empirical Test of the Utility of the Observations-To-Variables Ratio in Factor and Components Analysis. *Applied Psychological Measurement* 9, 2 (1985), 165–178. DOI: http://dx.doi.org/10.1177/014662168500900205

[4] Hala Assal and Sonia Chiasson. 2019. 'Think Secure from the Beginning': A Survey with Software Developers. In *Proceedings of the 37th CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 289, 13 pages. DOI: http://dx.doi.org/10.1145/3290605.3300519

[5] Andrew R Baggaley. 1983. Deciding on the ratio of number of subjects to number of variables in factor analysis. *Multivariate Experimental Clinical Research* (1983).

[6] Richard P. Bagozzi and Youjae Yi. 1988. On the evaluation of structural equation models. *Journal of the Academy of Marketing Science* 16, 1 (01 Mar 1988), 74–94. DOI:http://dx.doi.org/10.1007/BF02723327

[7] Albert Bandura. 1993. Perceived Self-Efficacy in Cognitive Development and Functioning. *Educational Psychologist* 28, 2 (1993), 117–148. DOI: http://dx.doi.org/10.1207/s15326985ep2802_3

[8] Paul T Barrett and Paul Kline. 1981. The observation to variable ratio in factor analysis. *Personality Study & Group Behaviour* (1981).

[9] Peter M Bentler. 1990. Comparative fit indexes in structural models. *Psychological bulletin* 107, 2 (1990), 238.

[10] Peter M Bentler and Douglas G Bonett. 1980. Significance tests and goodness of fit in the analysis of covariance structures. *Psychological bulletin* 88, 3 (1980), 588.

[11] Kevin Bock, George Hughey, and Dave Levin. 2018. King of the Hill: A Novel Cybersecurity Competition for Teaching Penetration Testing. In *Proceedings of the 3rd USENIX Workshop on Advances in Security Education (ASE '18)*. USENIX Association, Baltimore, MD. https://www.usenix.org/conference/ase18/presentation/bock

[12] Kenneth A Bollen. 2014. *Structural equations with latent variables*. Vol. 210. John Wiley & Sons.

[13] Richard W Brislin. 1980. Cross-cultural research methods. In *Environment and culture*. Springer, 47–82.

[14] John T. Cacioppo, Richard E. Petty, and Chuan Feng Kao. 1984. The Efficient Assessment of Need for Cognition. *Journal of Personality Assessment* 48, 3 (1984), 306–307. DOI: http://dx.doi.org/10.1207/s15327752jpa4803_13 PMID: 16367530.

[15] A Colin Cameron and Pravin K Trivedi. 2013. *Regression analysis of count data*. Vol. 53. Cambridge university press.

[16] Edward G Carmines and John McIver. 1981. Analyzing models with unobserved variables. *Social measurement: Current issues* 80 (1981).

[17] Raymond B. Cattell. 1952. *Factor analysis: An introduction and manual for the psychologist and social scientist*. Harper & Row, New York.

[18] Center for Cyber Safety and Education. 2017. *Global Information Security Workforce Study*. Technical Report. Center for Cyber Safety and Education, Clearwater, FL. https://iamcybersafe.org/wp-content/uploads/2017/07/N-America-GISWS-Report.pdf

[19] Barbara A. Cerny and Henry F. Kaiser. 1977. A Study Of A Measure Of Sampling Adequacy For Factor-Analytic Correlation Matrices. *Multivariate Behavioral Research* 12, 1 (1977), 43–47. DOI: http://dx.doi.org/10.1207/s15327906mbr1201_3 PMID: 26804143.

[20] Pravir Chandra. 2017. *Software Assurance Maturity Model*. Technical Report. Open Web Application Security Project.

[21] Wu chang Feng, Robert Liebman, Lois Delcambre, Michael Lupro, Tim Sheard, Scott Britell, and Gerald Recktenwald. 2017. CyberPDX: A Camp for Broadening Participation in Cybersecurity. In

*Proceedings of the 2nd USENIX Workshop on Advances in Security Education (ASE '17)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/ase17/workshop-program/presentation/feng

[22] Peter Chapman, Jonathan Burket, and David Brumley. 2014. PicoCTF: A Game-Based Computer Security Competition for High School Students. In *Proceedings of the 1st USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE '14)*. USENIX Association, San Diego, CA. http://www.usenix.org/conference/3gse14/summit-program/presentation/chapman

[23] Kevin Chung and Julian Cohen. 2014. Learning Obstacles in the Capture The Flag Model. In *Proceedings of the 1st USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE '14)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/3gse14/summit-program/presentation/chung

[24] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[25] Andrew L Comrey. 1973. *A first course in factor analysis*. Academic Press, New York.

[26] Andrew L Comrey. 1978. Common methodological problems in factor analytic studies. *Journal of consulting and clinical psychology* 46, 4 (1978), 648.

[27] Douglas P Crowne and David Marlowe. 1960. A new scale of social desirability independent of psychopathology. *Journal of consulting psychology* 24, 4 (1960), 349.

[28] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. 2018. Investigating System Operators' Perspective on Security Misconfigurations. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS '18)*. ACM.

[29] Wenliang Du. 2011. SEED: hands-on lab exercises for computer security education. *IEEE Security & Privacy* 9, 5 (2011), 70–73.

[30] C. Eagle. 2013. Computer Security Competitions: Expanding Educational Outcomes. *IEEE Security Privacy* 11, 4 (July 2013), 69–71. DOI: http://dx.doi.org/10.1109/MSP.2013.83

[31] Serge Egelman and Eyal Peer. 2015. Scaling the Security Wall: Developing a Security Behavior Intentions Scale (SeBIS). In *Proceedings of the 33rd CHI Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 2873–2882. DOI: http://dx.doi.org/10.1145/2702123.2702249

[32] Brian S Everitt and Anders Skrondal. 2010. *The Cambridge dictionary of statistics*. New York University.

[33] Leandre R Fabrigar, Duane T Wegener, Robert C MacCallum, and Erin J Strahan. 1999. Evaluating the use of exploratory factor analysis in psychological research. *Psychological methods* 4, 3 (1999), 272.

[34] Cori Faklaris, Laura A. Dabbish, and Jason I. Hong. 2019. A Self-Report Measure of End-User Security Attitudes (SA-6). In *Proceedings of the 5th Symposium on Usable Privacy and Security (SOUPS '19)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/soups2019/presentation/faklaris

[35] The Association for Computing Machinery. 2019. Chapters. https://acm.org/chapters. (2019).

[36] Claes Fornell and David F. Larcker. 1981. Evaluating Structural Equation Models with Unobservable Variables and Measurement Error. *Journal of Marketing Research* 18, 1 (1981), 39–50. DOI: http://dx.doi.org/10.1177/002224378101800104

[37] Gordon Fraser, Alessio Gambi, Marvin Kreis, and José Miguel Rojas. 2019. Gamifying a Software Testing Course with Code Defenders. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 571–577. DOI: http://dx.doi.org/10.1145/3287324.3287471

[38] Richard L. Gorsuch. 1978. *Factor Analysis* (2nd ed.). Erlbaum, Hillsdale, NJ.

[39] Richard L. Gorsuch. 1988. *Exploratory Factor Analysis*. Springer US, Boston, MA, 231–258. DOI: http://dx.doi.org/10.1007/978-1-4613-0893-5_6

[40] Edward Guadagnoli and Wayne F Velicer. 1988. Relation of sample size to the stability of component patterns. *Psychological bulletin* 103, 2 (1988), 265.

[41] Joy Paul Guilford. 1954. Psychometric methods. (1954).

[42] Joseph F Hair, William C Black, Barry J Babin, Rolph E Anderson, Ronald L Tatham, and others. 2006. Multivariate data analysis. (2006).

[43] Mariana Hentea, Harpal S Dhillon, and Manpreet Dhillon. 2006. Towards changes in information security education. *Journal of Information Technology Education: Research* 5 (2006), 221–233.

[44] Timothy R. Hinkin, J. Bruce Tracey, and Cathy A. Enz. 1997. Scale Construction: Developing Reliable and Valid Measurement Instruments. *Journal of Hospitality & Tourism Research* 21, 1 (1997), 100–120. DOI: http://dx.doi.org/10.1177/109634809702100108

[45] Sture Holm. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics* 6, 2 (1979), 65–70. http://www.jstor.org/stable/4615733

[46] John L. Horn. 1965. A rationale and test for the number of factors in factor analysis. *Psychometrika* 30, 2 (01 Jun 1965), 179–185. DOI: http://dx.doi.org/10.1007/BF02289447

[47] Gary Hsieh and Rafal Kocielnik. 2016. You Get Who You Pay for: The Impact of Incentives on Participation Bias. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social*

*Computing (CSCW '16)*. ACM, New York, NY, USA, 823–835. DOI: http://dx.doi.org/10.1145/2818048.2819936

[48] Luigi Lo Iacono and Peter Leo Gorski. 2017. I Do and I Understand. Not Yet True for Security APIs. So Sad. In *Proceedings of the 2nd European Workshop on Usable Security (EuroUSEC '17)*. Internet Society. https://doi.org/10.14722/eurousec

[49] Ge Jin, Manghui Tu, Tae-Hoon Kim, Justin Heffron, and Jonathan White. 2018. Game Based Cybersecurity Training for High School Students. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 68–73. DOI: http://dx.doi.org/10.1145/3159450.3159591

[50] 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 35th International Conference on Software Engineering (ICSE '13)*. IEEE Press, 672–681.

[51] Melanie Jones. 2019. Why cybersecurity education matters. https://www.itproportal.com/features/why-cybersecurity-education-matters/. (2019).

[52] Karl G Jöreskog and Dag Sörbom. 1993. *LISREL 8: Structural equation modeling with the SIMPLIS command language*. Scientific Software International.

[53] Lindsay Kolowich. 2017. The Demographics of Developers Around the World. https://blog.hubspot.com/marketing/developers-demographic-survey. (2017).

[54] Stefan Krüger, Johannes Späth, Karim Ali, Eric Bodden, and Mira Mezini. 2017. CrySL: Validating Correct Usage of Cryptographic APIs. *CoRR* abs/1710.00564 (2017). http://arxiv.org/abs/1710.00564

[55] Joseph T. Kunce, Daniel W. Cook, and Douglas E. Miller. 1975. Random Variables and Correlational Overkill. *Educational and Psychological Measurement* 35, 3 (1975), 529–534. DOI: http://dx.doi.org/10.1177/001316447503500301

[56] Ákos Lédeczi, MiklÓs MarÓti, Hamid Zare, Bernard Yett, Nicole Hutchins, Brian Broll, Péter Völgyesi, Michael B. Smith, Timothy Darrah, Mary Metelko, Xenofon Koutsoukos, and Gautam Biswas. 2019. Teaching Cybersecurity with Networked Robots. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. ACM, New York, NY, USA, 885–891. DOI: http://dx.doi.org/10.1145/3287324.3287450

[57] Timothy C Lethbridge, Jorge Diaz-Herrera, Richard Jr J LeBlanc, and J Barrie Thompson. 2007. Improving software practice through education: Challenges and future trends. In *Future of Software Engineering*. IEEE Computer Society, 12–28.

[58] Richard Harold Lindeman. 1980. *Introduction to bivariate and multivariate analysis*. Technical Report.

[59] LinkedIn. 2019. LinkedIn. https://linkedin.com/. (2019).

[60] J Scott Long. 1983. *Confirmatory factor analysis: A preface to LISREL*. Vol. 33. Sage Publications.

[61] Robert Loo. 1983. Caveat on Sample Sizes in Factor Analysis. *Perceptual and Motor Skills* 56, 2 (1983), 371–374. DOI: http://dx.doi.org/10.2466/pms.1983.56.2.371

[62] Robert C MacCallum, Michael W Browne, and Hazuki M Sugawara. 1996. Power analysis and determination of sample size for covariance structure modeling. *Psychological methods* 1, 2 (1996), 130.

[63] LA Maruscuilo and JR Levin. 1983. Multivariate statistics in the social sciences. *Books/Cole, Monterrey, California* (1983).

[64] Gary McGraw, Sammy Migues, and Jacob West. 2018. *Building Security in Maturity Model*. Technical Report. Open Web Application Security Project.

[65] Robert K McKinley, Terjinder Manku-Scott, Adrian M Hastings, David P French, and Richard Baker. 1997. Reliability and validity of a new measure of patient satisfaction with out of hours primary medical care in the united kingdom: development of a patient questionnaire. *BMJ* 314, 7075 (1997), 193. DOI: http://dx.doi.org/10.1136/bmj.314.7075.193

[66] Adam W Meade and S Bartholomew Craig. 2012. Identifying careless responses in survey data. *Psychological methods* 17, 3 (2012), 437.

[67] Meetup. 2019. We are what we do | Meetup. https://www.meetup.com/. (2019).

[68] Microsoft. 2019. Microsoft Security Development Lifecycle Practices. https://www.microsoft.com/en-us/securityengineering/sdl/practices. (2019).

[69] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 37th CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 140, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300370

[70] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 311–328. DOI: http://dx.doi.org/10.1145/3133956.3134082

[71] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. 2018. Deception Task Design in Developer Password Studies: Exploring a Student Sample. In *Proceedings of the 14th Symposium*

*on Usable Privacy and Security (SOUPS '18)*. USENIX Association, Baltimore, MD, 297–313. https://www.usenix.org/conference/soups2018/presentation/naiakshina

[72] Richard G Netemeyer, William O Bearden, and Subhash Sharma. 2003. *Scaling procedures: Issues and applications*. Sage Publications.

[73] William Newhouse, Stephanie Keith, Benjamin Scribner, and Greg Witte. 2017. *NIST Special Publication 800-181, The NICE Cybersecurity Workforce Framework*. Technical Report. National Institute of Standards and Technology.

[74] Jum C Nunnally. 1994. *Psychometric theory* (3rd ed.). Tata McGraw-Hill Education.

[75] National Institute of Standards and Technology. 2019. Software and Supply Chain Assurance Forum - Cyber Supply Chain Risk Management | CSRC. https://csrc.nist.gov/Projects/Supply-Chain-Risk-Management/SSCA. (2019).

[76] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. API Blindspots: Why Experienced Developers Write Vulnerable Code. In *Proceedings of the 14th Symposium on Usable Privacy and Security (SOUPS '18)*. USENIX Association, Baltimore, MD, 315–328. https://www.usenix.org/conference/soups2018/presentation/oliveira

[77] Eyal Peer, Laura Brandimarte, Sonam Samat, and Alessandro Acquisti. 2017. Beyond the Turk: Alternative platforms for crowdsourcing behavioral research. *Journal of Experimental Social Psychology* 70 (2017), 153 – 163. DOI:http://dx.doi.org/https://doi.org/10.1016/j.jesp.2017.01.006

[78] K. Qian, D. Lo, H. Shahriar, L. Li, F. Wu, and P. Bhattacharya. 2017. Learning database security with hands-on mobile labs. In *2017 IEEE Frontiers in Education Conference (FIE)*. 1–6. DOI:http://dx.doi.org/10.1109/FIE.2017.8190716

[79] Adrian E Raftery. 1995. Bayesian model selection in social research. *Sociological methodology* (1995), 111–163.

[80] Gilles Raîche, Theodore A. Walls, David Magis, Martin Riopel, and Jean-Guy Blais. 2013. Non-Graphical Solutions for Cattell's Scree Test. *Methodology* 9, 1 (2013), 23–29. DOI:http://dx.doi.org/10.1027/1614-2241/a000051

[81] Prashanth Rajivan, Pablo Moriano, Timothy Kelley, and Linda Jean Camp. 2016. What Can Johnny Do?–Factors in an End-User Expertise Instrument. In *Proceedings of the 10th International Symposium on Human Aspects of Information Security and Assurance (HAISA '16)*. Frankfurt, Germany, 199–208.

[82] Tenko Raykov. 1997. Scale Reliability, Cronbach's Coefficient Alpha, and Violations of Essential Tau-Equivalence with Fixed Congeneric Components. *Multivariate Behavioral Research* 32, 4 (1997), 329–353. DOI:http://dx.doi.org/10.1207/s15327906mbr3204_2 PMID: 26777071.

[83] Tony Rice, Josh Brown-White, Tania Skinner, Nick Ozmore, Nazira Carlage, Wendy Poland, Eric Heitzman, and Danny Dhillon. 2018. *Fundamental Practices for Secure Software Development*. Technical Report. Software Assurance Forum for Excellence in Code.

[84] John P Robinson, Phillip R Shaver, and Lawrence S Wrightsman. 1991. Criteria for scale selection and evaluation. *Measures of personality and social psychological attitudes* 1, 3 (1991), 1–16.

[85] Dale C. Rowe, Barry M. Lunt, and Joseph J. Ekstrom. 2011. The Role of Cyber-security in Information Technology Education. In *Proceedings of the 12th Conference on Information Technology Education (SIGITE '11)*. ACM, New York, NY, USA, 113–122. DOI:http://dx.doi.org/10.1145/2047594.2047628

[86] Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L. Mazurek, and Piotr Mardziel. 2016. Build It, Break It, Fix It: Contesting Secure Development. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 690–703. DOI:http://dx.doi.org/10.1145/2976749.2978382

[87] 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 (ICSE '15)*. IEEE Press, 598–608.

[88] Ralf Schwarzer, Matthias Jerusalem, J Weinman, S Wright, and M Johnston. 1995. Measures in health psychology: A user?s portfolio. Causal and control beliefs. *Generalized Self-Efficacy Scal, NFER-NELSON, Windsor* (1995), 35–37.

[89] Yan Shoshitaishvili, Michael Weissbacher, Lukas Dresel, Christopher Salls, Ruoyu Wang, Christopher Kruegel, and Giovanni Vigna. 2017. Rise of the HaCRS: Augmenting Autonomous Cyber Reasoning Systems with Human Assistance. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, Dallas, Texas, USA, 347–362. DOI:http://dx.doi.org/10.1145/3133956.3134105

[90] 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 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '15)*. ACM, 248–259. DOI:http://dx.doi.org/10.1145/2786805.2786812

[91] George W Snedecor and Witiiam G Cochran. 1989. Statistical methods. *Ames: Iowa State Univ. Press Iowa* (1989).

[92] StackOverflow. 2018. StackOverflow Developer Survey Results 2018. https://insights.stackoverflow.com/survey/2018. (2018).

[93] Barbara G Tabachnick, Linda S Fidell, and Jodie B Ullman. 2007. *Using multivariate statistics*. Vol. 5. Pearson Boston, MA.

[94] 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 *Proceedings of the 2nd Workshop on Security Information Workers (WSIW '16)*. USENIX Association, Denver, CO. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/thomas

[95] Meng-Jung Tsai, Ching-Yeh Wang, and Po-Fen Hsu. 2019. Developing the Computer Programming Self-Efficacy Scale for Computer Literacy Education. *Journal of Educational Computing Research* 56, 8 (2019), 1345–1360. DOI: http://dx.doi.org/10.1177/0735633117746747

[96] Li tze Hu and Peter M. Bentler. 1999. Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling: A Multidisciplinary Journal* 6, 1 (1999), 1–55. DOI: http://dx.doi.org/10.1080/10705519909540118

[97] Upworki. 2017. Hire Freelancers, Make Things Happen | Upwork. https://upwork.com. (2017).

[98] Wayne F. Velicer, Andrew C. Peacock, and Douglas N. Jackson. 1982. A Comparison Of Component And Factor Patterns: A Monte Carlo Approach. *Multivariate Behavioral Research* 17, 3 (1982), 371–388. DOI: http://dx.doi.org/10.1207/s15327906mbr1703_5 PMID: 26800757.

[99] Daniel Votipka, Rock Stevens, Elissa M. Redmiles, Jeremy Hu, and Michelle L. Mazurek. 2018. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. In *Proceedings of the 39th IEEE Symposium on Security and Privacy (SP '18)*. San Francisco, CA, 374–391. DOI: http://dx.doi.org/10.1109/SP.2018.00003

[100] Stacey Watson and Heather Richter Lipford. 2017. A Proposed Visualization for Vulnerability Scan Data. In *Proceedings of the 3rd Workshop on Security Information Workers (WSIW '17)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/soups2017/workshop-program/wsiw2017/watson

[101] Susan C Weller and A Kimball Romney. 1988. *Systematic data collection*. Vol. 10. Sage publications.

[102] Joseph Werther, Michael Zhivich, Tim Leek, and Nickolai Zeldovich. 2011. Experiences in Cyber Security Education: The MIT Lincoln Laboratory Capture-the-flag Exercise. In *Proc. of the 4th Conference on Cyber Security Experimentation and Test (CSET'11)*. USENIX Association, Berkeley, CA, USA, 12–12. http://dl.acm.org/citation.cfm?id=2027999.2028011

[103] Chamila Wijayarathna and Nalin A. G. Arachchilage. 2018. Why Johnny Can't Store Passwords Securely?: A Usability Evaluation of Bouncycastle Password Hashing. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (EASE'18)*. ACM, New York, NY, USA, 205–210. DOI: http://dx.doi.org/10.1145/3210459.3210483

[104] 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 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '15)*. ACM, 260–271. DOI: http://dx.doi.org/10.1145/2786805.2786816

[105] Irene M.Y. Woon and Atreyi Kankanhalli. 2007. Investigation of IS professionals' intention to practise secure development of applications. *International Journal of Human-Computer Studies* 65, 1 (2007), 29 – 41. DOI: http://dx.doi.org/https://doi.org/10.1016/j.ijhcs.2006.08.003

## EXPERT REVIEW SURVEY

### Instructions

Thank you for participating as an expert reviewer. Throughout this survey, you will be shown a list of secure software development tasks we have collected from several secure development guidelines and frameworks including NIST's NICE framework, Microsoft's SDL, BSIMMs, and OpenSAMM.
For each secure development task, you will be asked to indicate whether you believe the task is or is not actually a component of the secure development process. Specifically, we care about tasks that have an effect on the end product software. These tasks could be carried out by any member of the organization (e.g., programmer, product manager, etc) at any phase of the development process (e.g., requirements, design, implementation, etc), but must have an effect on the final code that is produced.
Through your review, we hope to remove any items from our list that may have been incorrectly included. Additionally, we will ask if the task wording is clear. If you find the text to be unclear or confusing, please briefly explain your confusion so that we can improve the statement's clarity.
If there are any secure development tasks that you do not believe were covered, please provide these at the conclusion of the survey so that they can be added to our list.
Note, for this survey we ask that you consider each task listed independently. For example, multiple tasks may reflect similar concepts. Please consider the adequateness of each of these tasks separately. Additionally, the given order the tasks is arbitrary. They are not meant to represent any sequential order of operations, but instead are tasks that should be completed at some point in the secure development process (in some cases multiple times).

### Tasks

[Task description from Table 1]

1. Select the option below that best reflects the above task.
   (a) Definitely a secure development task
   (b) Probably a secure development task
   (c) Probably not a secure development task
   (d) Definitely not a secure development task
   (e) Unsure
2. Did you find the text for the above task unclear or confusing?
   (a) Yes. Please briefly explain what is confusing or unclear.
   (b) No

### Final Thoughts

1. Are there any secure software development tasks that we did not include? Please list all such tasks below, each on its own line.
2. Do you have any other questions, comments, concerns, or suggestions for our research team that were not addressed throughout the survey?

### Demographics

1. Please specify your current (or most recent) job title.
2. How many total years of experience do you have with secure software development?
3. Please specify the highest degree or level of school you have completed
   (a) Some high school credit, no diploma
   (b) High school graduate, diploma or equivalent (e.g., GED)
   (c) Some college credit, no degree
   (d) Associate's degree
   (e) Bachelor's degree
   (f) Master's degree
   (g) Doctoral degree
4. If you are currently a student or have completed a college degree, please specify your field(s) of study (e.g. Biology, Computer Science, etc).

## MAIN SURVEY

### Instructions

Throughout this survey, you will be shown a list of software development tasks. For each software development task, you will be asked to indicate your confidence in your ability to complete the task.

Note that throughout we refer to your ability to perform each task with respect to the system under development. Here, when we refer to the "system", we specifically mean the features you are involved in the design and implementation of. This could be a single program or collection of programs. For example, If you work on a specific set of features or an application that integrates with a large environment of programs, the "system" would only be those features or applications you work on.

After indicating your confidence in completing each task, you will be asked to answer a few additional questions regarding your decision-making.

Finally, you will be asked to answer a few demographic questions.

*(Note: Only participants in the fourth step of the scale development process were asked to answer additional scale questions)*

### Tasks

Please rate your level of confidence in completing the software development tasks given below.

["I can" statements from Table 2]

1. I am not confident at all
2. I am slightly confident
3. I am somewhat confident
4. I am moderately confident
5. I am absolutely confident
6. I do not understand the question

**Demographics**

1. How many total years of experience do you have with software development?
2. Have you ever identified a security error in yours or someone else's code?
   (a) Never
   (b) Once
   (c) Multiple times
3. Has anyone ever identified a security error in code you have written?
   (a) Never
   (b) Once
   (c) Multiple times
4. Have you ever worked with someone you would consider a secure development expert?
   (a) Never
   (b) I have worked with one security expert
   (c) I have worked with multiple security experts
5. How often do you communicate with security experts?
   (a) Yearly
   (b) Quarterly
   (c) Monthly
   (d) Weekly
   (e) Daily
6. Have you ever discussed potential security problems with a secure development expert?
   (a) Never
   (b) Once
   (c) A few times
   (d) On a regular basis
7. Please select the range that most closely matches the amount of time you typically spend performing software development tasks per week.
   (a) < 5 hours
   (b) 5-10 hours
   (c) 10-20 hours
   (d) 20-30 hours
   (e) 30-40 hours
   (f) > 40 hours
8. Please indicate the response below that most closely matches your software development skill level:
   (a) Expert (recognized authority)
   (b) Advanced (applied theory)
   (c) Intermediate (practical application)
   (d) Novice (limited experience)
   (e) Fundamental awareness (basic knowledge)
   (f) Not applicable
9. Have you ever participated in any training related to software security (e.g., coursework, employer training, online exercises, etc)?
   (a) Yes
   (b) No
   (c) Not sure
10. Please select from the list of training types below, all the types of prior software security training you have participated in:
    (a) Academic coursework
    (b) Certification
    (c) Security conference
    (d) Online exercises
    (e) Employer training
11. Please specify the gender with which you most closely identify.
    (a) Male
    (b) Female
    (c) Other
    (d) Prefer not to answer
12. Please specify your age
    (a) 18-29
    (b) 30-39
    (c) 40-49
    (d) 50-59
    (e) 60-69
    (f) Over 70
13. Please specify your ethnicity. Select all that apply.
    (a) White
    (b) Hispanic or Latino
    (c) Black or African American
    (d) American Indian or Alaska Native
    (e) Asian, Native Hawaiian, or Pacific Islander
    (f) Native Hawaiian or Pacific Islander
    (g) Other
    (h) Prefer not to answer
14. Please specify the highest degree or level of school you have completed
    (a) Some high school credit, no diploma
    (b) High school graduate, diploma or equivalent (e.g., GED)
    (c) Some college credit, no degree
    (d) Associate's degree
    (e) Bachelor's degree
    (f) Master's degree
    (g) Doctoral degree
15. If you are currently a student or have completed a college degree, please specify your field(s) of study (e.g. Biology, Computer Science, etc).
16. Please select the response option that best describes your current employment status.
    (a) Working for payment or profit
    (b) Unemployed
    (c) Looking after home/family
    (d) Student

(e) Retired
(f) Unable to work due to permanent sickness or disability
(g) Other
(h) Prefer not to answer

17. Please specify the range which most closely matches your total, pre-tax, personal income specifically from software development last year.
(a) < \$999
(b) \$1,000-\$4,999
(c) \$5,000-\$14,999
(d) \$15,000-\$29,999
(e) \$30,000-\$49,999
(f) \$50,000-\$74,999
(g) \$75,000-\$99,999
(h) \$100,000-\$124,999
(i) \$125,000-\$149,999
(j) \$150,000-\$199,999
(k) > \$200,000
(l) Prefer not to asnwer

## ADDITIONAL DATA

In this appendix, we provide additional data that was not presented in the main paper for space considerations. Specifically, this includes the distribution of software development experience across participants assigned each additional scale and the observed correlations between SSD-SES and each additional scale.

| Scale | 1-2 years | 3-11 years | > 11 years |
|---|---|---|---|
| CPSES | 1 | 34 | 14 |
| SA-6 | 6 | 30 | 12 |
| SeBIS | 4 | 34 | 12 |
| GES | 5 | 33 | 6 |
| NFC | 6 | 30 | 13 |
| GDMS | 8 | 29 | 15 |
| Total | 15 | 95 | 36 |

Table 6: The distribution of participants' software development experience who were shown each additional scale.

| | Vulnerability Identification and Mitigation | Security Communication |
|---|---|---|
| **Convergent Validity** | | |
| CPSES | $\rho = 0.528$, p = 0.001 | $\rho = 0.627$, p < 0.001 |
| **Discriminant Validity** | | |
| SA-6 | $\rho = 0.249$, p = 0.191 | $\rho = 0.300$, p = 0.111 |
| $SeBIS_1$ | $\rho = 0.077$, p = 0.622 | $\rho = 0.202$, p = 0.298 |
| $SeBIS_2$ | $\rho = 0.370$, p = 0.053 | $\rho = 0.176$, p = 0.337 |
| $SeBIS_3$ | $\rho = 0.308$, p = 0.096 | $\rho = 0.227$, p = 0.224 |
| $SeBIS_4$ | $\rho = 0.145$, p = 0.412 | $\rho = 0.122$, p = 0.470 |
| GES | $\rho = 0.363$, p = 0.067 | $\rho = 0.375$, p = 0.063 |
| **Other Psychometric Scales** | | |
| NFC | $\rho = 0.464$, p = 0.007 | $\rho = 0.183$, p = 0.337 |
| $GDMS_1$ | $\rho = 0.013$, p = 0.928 | $\rho = 0.129$, p = 0.450 |
| $GDMS_2$ | $\rho = 0.276$, p = 0.121 | $\rho = 0.178$, p = 0.337 |
| $GDMS_3$ | $\rho = 0.148$, p = 0.403 | $\rho = -0.075$, p = 0.622 |
| $GDMS_4$ | $\rho = -0.075$, p = 0.622 | $\rho = 0.139$, p = 0.403 |
| $GDMS_5$ | $\rho = 0.271$, p = 0.121 | $\rho = 0.307$, p = 0.096 |

Table 7: Correlations between sub-scales, related scales, and other psychometrics.