Barriers to Shift-Left Security: The Unique Pain Points of Writing Automated Tests Involving Security Controls

Danielle Gonzalez
Rochester Institute of Technology
Rochester, NY, USA
dng2551@rit.edu

Paola Peralta Perez
Rochester Institute of Technology
Rochester, NY, USA
php5185@rit.edu

Mehdi Mirakhorli
Rochester Institute of Technology
Rochester, NY, USA
mxmvse@rit.edu

ABSTRACT

Background: Automated unit and integration tests allow software development teams to continuously evaluate their application's behavior and ensure requirements are satisfied. Interest in explicitly testing *security* at the unit and integration levels has risen as more teams begin to shift security left in their workflows, but there is little insight into any potential pain points developers may experience as they learn to adapt their existing skills to write these tests. **Aims**: Identify security unit and integration testing pain points that could negatively impact efforts to shift security (testing) left to this level. **Method**: An mixed-method empirical study was conducted on 525 Stack Overflow and Security Stack Exchange posts related to security unit and integration testing. Latent Dirichlet Allocation (LDA) was applied to identify commonly discussed topics, pain points were learned through qualitative analysis, and links were analyzed to study commonly-shared resources. **Results**: Nine topics representing security controls, components, and scenarios were identified; *Authentication* was the most commonly tested control. Developers experienced seven pain points unique to security unit and integration testing, which were all influenced by the complexity of security control designs and implementations. Most linked resources were other Q&A posts, but repositories and documentation for security tools and libraries were also common.**Conclusions**: Developers may experience several unique pain points when writing tests at this level involving security controls. Additional resources are needed to guide developers through these challenges, which should also influence the creation of strategies and tools to help shift security testing to this level. To accelerate this, actionable recommendations for practitioners and future research directions based on these findings are highlighted.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; • **Security and privacy → Software security engineering**.

KEYWORDS

Shift-Left Security, Unit Testing, Integration Testing, Pain Points, Security Testing, Latent Dirichlet Allocation, Stack Overflow

ACM Reference Format:
Danielle Gonzalez, Paola Peralta Perez, and Mehdi Mirakhorli. 2021. Barriers to Shift-Left Security: The Unique Pain Points of Writing Automated Tests Involving Security Controls. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21), October 11–15, 2021, Bari, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3475716.3475786

1 INTRODUCTION

Software development teams use automated unit and integration tests to continuously evaluate low-level system functionality during implementation. This type of developer-driven testing is now a standard (or at least familiar) practice in most development workflows and is a key element of iterative process models like Extreme Programming (XP), Agile, Test Driven Development (TDD), Behavior Driven Development (BDD), and DevOps [? ? ? ?].

The need to 'shift security left' has motivated efforts to adapt these process models by integrating automated security testing into developer workflows, including at the unit and integration levels [?]. However, for developers this is a non-traditional application of infrastructure and practices they are already familiar with and presently there is little insight into any potential barriers that would impede explicitly security-focused testing at this level. Security, as a non-functional quality attribute, has traditionally been tested late in the development lifecycle. In many cases, tests are executed manually on fully-functional system instances by security experts with no involvement in their design or construction [?]. OWASP guidelines [? ?] recommend writing security unit and integration tests but provide little concrete advice for doing so. This motivated one prior study that analyzed authentication unit and integration tests to create a test planning guide [?], but associated pain points were not investigated. These challenges must influence the creation or enhancement of strategies and guidelines, as there is already an observed gap between available security resources for developers and the tasks they struggle with [? ? ?]. Developers seek problem solving help from their peers on Q&A websites [?], making them a good data source to study pain points.

This paper presents a mixed-method empirical study wherein 525 Q&A posts about security unit and integration testing were mined from Stack Overflow and Security Stack Exchange and analyzed using quantitative (LDA) and qualitative (coding) methods to learn (1) which security controls, scenarios, and controls are being discussed and tested by developers, (2) what pain points they experience when writing these tests, and (3) any resources developers

commonly reference in their questions or or recommend in answers. To collect sufficient data to investigate the above concerns, three independent analyses were conducted, influenced by the successes and limitations of prior studies using Q&A data to study challenges associated with specific concerns [? ? ? ?].
Accordingly, this work was driven by three research questions:
RQ1: What are the key topics discussed in Q&A posts about unit and integration testing security controls? Topic modeling is a popular technique for Q&A studies that allows extraction and definition of topic categories from large corpora of text data that would be otherwise unreasonable to analyze manually. The motivation to identify topics for this study is twofold. First, prior studies (Section 2.2) show this approach is well-suited to identifying key discussion topics related to developer challenges. Second, this technique can be applied to the entire dataset, meaning the topics can be used to reason about findings from smaller qualitative analyses. The topics identified in this study represent commonly-tested security controls, scenarios, and components.
RQ2: What security unit and integration testing pain points do developers ask for help with? To identify pain points which may be barriers to adoption of security unit and integration testing, 50 of the top Q&A posts collected were manually analyzed using open coding [?]. Conducting such in-depth analysis is critical to achieve the goal identifying pain points unique to security-related tests enriched with sufficient contextual data to drive efforts to address them.
RQ3: What type of resources do developers link to in the discussions of these questions? Developers often share links to resources they have referenced when posting questions and to recommend potentially helpful information sources in their answers. To investigate what resources are commonly shared in security unit and integration testing posts, an exploratory analysis was conducted to identify the most frequently shared resources and a randomly selected set of links from each top domain were manually reviewed to investigate covered topics.

Topic-based studies such as this demonstrate how Q&A data can be used to identify specific technical concerns developers face relating to security, and these insights can help researchers maintain relevancy with their work.

2 BACKGROUND & RELATED WORK

This section provides background and related work on security unit testing and empirical studies of Q&A posts.

2.1 (Security) Unit and Integration Testing

Unit and integration tests are written by developers to evaluate system behavior at different levels during implementation. As their names imply, unit tests focus on the behavior of isolated 'units' (classes or methods) in the code, while integration tests verify interactions between multiple units or components [?]. A suite (collection) of unit and integration tests can be executed *automatically* as part of a continuous integration (CI) pipeline to test changes as they are submitted or *manually* by a developer to test changes as they are implemented. In practice, the distinction between unit and integration tests can be unclear; most automated testing frameworks can be used to write and execute both, and defect detection performance has been shown to be similar in prior studies [? ?]. Developers may not care about this distinction in practice [?] and do not always specify the intended level in their Q&A posts.
Testing Security at the Unit and Integration Level As noted in Section 1, little academic work has focused on the practice of explicitly testing security controls at this level. However, this practice is encouraged by respected security organizations [? ?] and practitioners [? ? ?] although actionable guidelines for use in practice are lacking. From the technical perspective, Mohammadi *et al.* have proposed approaches for writing tests to detect cross-site scripting (XSS) vulnerabilities [? ?]. Motivated by the discrepancy between recommendations and guidelines for security unit testing and the lack of insight into what security control scenarios developers test at this level, Gonzalez *et al.* [?] mined 481 JUnit tests related to token authentication implemented using the Spring Security framework from 53 open source Java projects. They conducted a manual grounded theory analysis of these tests and developed a unit testing guide detailing the key elements comprising 53 unique test cases. However, they did not investigate any challenges developers may have faced when writing these tests. To the author's knowledge, the study described in this paper is the first to explore pain points associated with security unit and integration testing.

2.2 Stack Overflow & Security Stack Exchange

Stack Exchange is a network of specialized question and answer (Q&A) websites. Their flagship website is Stack Overflow (SO), an extremely popular site for computer programming Q&A and discussion[?]. Since its debut in 2008, more than 21 million questions have been posted to Stack Overflow, and visiting the site is integrated into the problem solving processes of many developers [?]. Security Stack Exchange (SSE) is another popular Q&A community where developers and security engineers discuss topics related to Information Security [?]. This has made them popular amongst software engineering researchers , who have developed efficient and accurate techniques for mining, filtering and analyzing post data to study a diverse set of software development concerns and subjects. Meldrum *et al.*'s 2017 systematic mapping study ($n = 265$) [?] and Ahmad *et al.*'s 2018 literature review ($n = 166$) [?] have quantified the research community's growing interest in Stack Overflow, popular topics, and common analysis techniques. Barua *et al.* and Ye *et al.* have conducted broad studies using *all* posts on Stack Overflow to explore key discussion topics [?] and the structure and dynamics of Stack Overflow from a knowledge network perspective [?]. Other works have used Stack Overflow to study a wide variety of software development topics including but not limited to: cloud computer vision [?], mobile development [? ? ?], software testing [?], software architecture [?], web development [?], Apache Spark [?], Java [?], Docker [?], and deep learning [?].
Security Topics & Challenges: Several studies have used Stack Overflow to study developer's discussion topics and challenges related to security [? ? ? ?]. Unlike this work, these studies did not include data from Security Stack Exchange. Yang *et al.* [?] wanted to investigate what *security*-related topics developers ask questions about; to identify relevant posts, they used two tag-based heuristics. An automated approach was used to analyze and group posts by topic, specifically feature modeling and LDA. As a result they

identified 30 topics, and investigated the distribution of posts to topic, most popular topic, and which topics were most difficult to answer questions about. From these insights the authors suggest researchers should investigate assistive solutions for the most popular and difficult topics, educators should focus on web security, and practitioners could use the difficulty metrics to assign topic-based work by level of expertise.

Human factors relating to secure software development has become a popular research area, as we seek to understand how developers perceive, respond to, and overcome security challenges [? ?]. Since Stack Overflow is widely-used by developers when faced with technical challenges, Lopez *et al.* [? ?] conducted two studies which applied qualitative methods (coding) to analyze a small sample (20) of highly-rated security posts and their answers to explore the social context of security knowledge sharing. Their first analysis revealed insights into three dimensions: *advice and assessment*, *values and attitudes*, and *community involvement*. They found that security awareness grows through discussing suggested solutions, and although most questions are technical, developers also discuss rationale, return-on-investment, balancing developers needs with external influences (regulations, organization-level requirements), and the effectiveness and circumstances of suggested solutions. Developers perspectives are often embedded in answers to technical questions, and the authors identified common themes: principles, trust, fear, pride, and responsibilities. Given the evolving nature of security, questions "belong to the community" and discussion activity can persist years after the original posting date. [?]. In their second study, they focused on finer-grained details of participation in the same set of questions, examining features of the discussion environment such as time, edits, quoting, and naming techniques. They observed differences in the question and answer comment streams; for example, answer comment streams focus more on technical details vs the perspectives found in question streams. [?]. While a small sample was used for these studies, their findings reveal the multi-dimensional nature of security discussions on Stack Overflow, and support the notion that both technical (challenges, concepts) and social (perspectives, opinions) factors relating to security topics can be learned by studying data from this platform.

Nadi *et al.*[?] manually analyzed 100 Stack Overflow posts as part of a study conducted to understand tasks developers perform with Java cryptography APIs, and the challenges they face. Their study was motivated by a need to understand what developers need help with in this regard, and which support mechanisms would be most effective. Findings from the post analysis were supplemented with two developer surveys (question askers and API-users) and an analysis of 100 GitHub repositories using Java cryptography APIs. They observed that posters with some domain knowledge struggled primarily with correct usage of the API, such as the correct sequence of method calls needed to perform a task. Also noted was posters lacking domain knowledge struggled with choosing the correct algorithm for their needs or which libraries to use. These findings are of particular interest to this work, as most security controls are implemented using third-party libraries, which might influence the questions asked about testing.

Testing Topics & Challenges: Key discussion topics and difficulties related to testing have been studied from the general perspective by Kochar *et al.* [?] and for Android development specifically by Villanes *et al.* [?]. Kochar *et al.*'s study used a similar mixed-method approach that combines LDA topic model insights with deeper qualitative analysis. They identified eight key testing topics from over 38,000 testing-related questions. Consistent with this work, *Login* and *Client/Server* were identified amongst others unrelated to security (*e.g.* Test Framework) in this topic list; they supplemented this with several smaller exploratory analysis about temporal trends, view-based topic popularity, and a special look at mobile testing concerns. They also manually reviewed 50 posts to identify challenges; these were associated directly to topics but show that similar to the pain points identified in this work, many relate to conceptual knowledge gaps and design confusions.
Link Sharing To identify what resources developers reference and recommend for security unit and integration testing challenges (RQ3), this study also includes an analysis of links shared in questions *and* answers. Liu *et al.* have conducted similar explorations of broken [?] and repeatedly shared links [?] on Stack Overflow. In 2020, Liu *et al.* presented a large-scale study of broken links wherein they report findings from testing the HTTP responses of over 12 million links from all posts created before June 2019. The authors then analyzed the 14.2% of links they classified as broken to examine their intended purpose and impact. Unlike [?], this study does not exclude "internal" links to other Q&A posts, but does use similar criteria to exclude "broken" links (Section 4.3). Liu *et al.* conducted an exploratory analysis [?] to investigate the characteristics of the repeatedly shared external links. Consistent with the findings of this study (Section 7) they found most common websites refer documentation and tutorials.

3 DATA COLLECTION

3.1 Mining Candidate Q&A Posts

For this work, we mined Q&A posts from Stack Overflow (SO) and Security Stack Exchange (SSE). As both sites are on the same platform, we were able to mine all the post data using the Stack Exchange API and Stack Exchange Data Explorer. One challenge that studies using Q&A data must address is searching and filtering for relevant posts. The standard approach used in prior work is to perform tag and content-based filtering, where keywords for the study topic are used to identify relevant posts [? ?]. To build a candidate set of posts from Stack Overflow, Le *et al.* [?]'s dataset of 97,051 security-related posts was used as the initial search space. Their collection is the largest to-date, and was curated using a novel automated learning framework designed to mitigate limitations of keyword and supervised filtering approaches. It was not necessary to filter post content for security relevance when searching Security Stack Exchange. Instead, the criteria used to identify candidate posts from this site was relevance to *unit* and *integration testing*. A query for posts with *'unit'* and *'integrat'* in their title, tags, or body yielded a candidate set of 1,310 posts.

3.2 Identifying Relevant Posts

Candidate posts from both sources then underwent a second filtering to identify posts from the SO set that were specifically related to unit or integration testing security controls and to confirm relevance for the posts mined from SSE. At this stage, we used a keyword-based approach to search the title, body, and tags of each post for these terms: *"unit-test","unittest", "unit-tests", "unittesting", "unit-testing", "unit test", "unit testing", "unit tests", "testing", "JUnit", "pyunit", "phpunit", "tdd", "bdd", "integration-test", "integration-testing", "integration test", "integration testing".*

These terms were chosen after manual experimentation with tag-based search on stack overflow. The three 'xUnit' terms (*"JUnit", "pyunit", "phpunit"*) were included because they were recommended by Stack Overflow as popular tags related to "unit-test". Several variations of the same terms are also considered to ensure that matches in tags (which use '-' for multiple terms) *and* title/body text would be captured. Regular expressions were used for the search to ensure that sub-strings were not considered matches. When a post has one or more matches, a record was also created mapping the matches to the field (*e.g.* title) they were found in. After manually reviewing preliminary results, posts that *only* had "test" or "testing" keywords in one of the three fields were removed because they were usually not related to unit or integration testing.
Final Dataset: After discarding posts unrelated to security unit and integration testing, the dataset consisted of 512 posts from Stack Overflow (SO) and 13 from Security Stack Exchange (SSE). As SSE focuses on information security, the small number of posts about testing at the unit and integration level from that source was expected. Combined, **the dataset used for this analysis consisted of 525 security unit and integration testing Q&A posts**. These posts were created between 2008 (when Stack Overflow began) and 2020, and 71% have an accepted answer. Table 1 provides other descriptive statistics for post scores, view counts, and answer counts.

Table 1: Descriptive Dataset Statistics

	Score	*# Views*	*# Answers*
Mean	2.09	1,673.24	1.43
Median	1	688	1
Mode	0	110	1
Range	87	38,317	9
Minimum	-4	17	0
Maximum	83	38,334	9

4 DATA ANALYSIS

To answer the research questions defined in Section 1, a structured mixed-method study was designed that combines elements of both quantitative (automated topic modeling) and qualitative (open coding) research. For each question an independent analysis was conducted, and the motivations and influences for this design are discussed in Section 1. The remainder of this section explains the specific methods and procedures used in each analysis.

4.1 Identification of Discussion Topics

To answer RQ1, the LDA topic modeling algorithm [?] was used to identify the most frequently-discussed topics in 525 security unit and integration testing Q&A posts. This section describes the text pre-processing pipeline applied to this data and the parameter tuning experiments conducted to train the topic model.

4.1.1 ***Text Pre-processing.*** The title and body content from each question was transformed into the appropriate representation for LDA using the following pipeline:

(1) Remove Links, Code Snippets and HTML Tags: These snippets were removed as this information is not relevant to the topic model [?]. To extract links, the text was searched for the tag. This data were preserved separately for use in further analysis (Section 4.3) for RQ3. Code snippets were identified by searching for the tag, and similarly general HTML tags (*e.g.* , ,) were also removed to reduce noise in the model.
(2) Remove Punctuation and Tokenize Text: After removing all punctuation characters, sentences were *tokenized* into lists of terms. The default approach is to isolate each individual word, but it is also possible to allow groups of words (*i.e.* n-grams) as a single term. During parameter tuning (see Section 4.1.2), bigrams (*e.g.* "unit test") and trigrams (*e.g.* "Latent Dirichlet Allocation") were considered, but the optimal configuration used single terms.
(3) Remove Stop Words: Common words and numbers that do not provide topic content (*e.g.*"the", "a","in") were removed.
(4) Normalize Terms: Terms were normalized to their canonical forms for consistency. This ensures that multiple forms of the same term (*e.g.* "configure","configuring", "configuration") are not considered unique terms. There are two approaches for normalization, lemmatization and stemming. Each produces a different result for the same word; for example "configuring" would be lemmatized as "configure" and stemmed to "configur". As discussed in Section 4.1.2, both approaches were considered during parameter tuning to account for influences on model performance and output readability.
(5) Frequency Filter: Filter extreme terms that appear in less than X documents or in more that Y% of all documents to reduce noise.

4.1.2 ***Topic Modeling.*** The Latent Dirichlet Allocation (LDA) algorithm [?] is commonly used for automated analysis of Stack Overflow posts through extraction of latent topics [? ? ? ? ? ? ? ? ? ?]. Likewise this technique was applied to extract topics from the set of 525 security unit and integration testing-related posts. An LDA model attempts to generate a pre-determined number of topics from a pre-processed corpus of text data. Like the 'number of topics' parameter, some choices made during pre-processing can influence the performance of the model. To determine the optimal topic model configuration for this data, we compared the performance of LDA models with 246 unique value combinations for the following 'parameters': term frequency filtering, term normalization, n-grams, and number of topics.
For this work the [?] implementation of LDA was used and its default values for the alpha and beta (word and topic density) hyperparameters were found to be sufficient in preliminary experiments. Standard coherence and perplexity metrics [? ?] were used to identify the optimal configuration for the dataset; these measures indicate the performance of an LDA model in terms of its ability to identify distinct but understandable topics. The optimal configuration removed terms found in less than 50 posts or more than 60% of all posts, used stemming to normalize terms, considered only 1-grams, and generated 10 topics.

4.1.3 ***Topic Labeling.*** Each 'topic' generated by an LDA model is represented by a list of terms and their weight (frequency) within the topic. Unique labels were assigned for each topic manually: two authors independently created label sets, which were compared, merged and revised until full agreement was reached.

4.2 Learning Developer's Pain Points

Developers post questions to Stack Overflow and other software-oriented Q&A websites when they are struggling to solve a conceptual (*"How do I do this?"*) or technical (*"Why isn't this working?"*) problem. It is therefore reasonable to expect that details about these problems could be extracted through topic modeling of post contents or analyzing tags added by post authors, but these data alone are insufficient for capturing important contextual details needed to learn the *pain points* behind these problems.

The LDA analysis conducted to answer RQ1 revealed that the key topics developers discussed (Section 5) in the security unit and integration testing posts examined represent commonly-tested security controls, scenarios, and components. This information was important towards understanding what developers were trying to test, but without details about how and why they struggled with these tests the resulting insights would be interesting but in-actionable. For example, no concrete recommendations can be made from a statement such as *"Developers find it difficult to test authentication at the unit and integration level."*. Adding tags did not improve this; tags in the posts studied reflected technical details (*e.g.* language) that provide no hints to problems and could impede efforts to identify language and domain-agnostic pain points.

Other studies [? ?] have mitigated these limitations by supplementing topic-based insights with qualitative analysis. Accordingly, a similar design was used for this study to identify security unit and integration testing pain points that are *associated with* but not strictly *dependent on* specific technical contexts or security controls. For example, one of the pain point identified (Section 6) was *'Creating HTTP Request Objects'*, which clearly correlates to the *HTTP Requests* topic (Section 5) identified by the LDA model. The deeper analysis enabled surface-level insights to be enriched with actionable details concerning *when*, *why*, and *how* writing related tests was difficult, such as: *"When testing login scenarios, developers struggled with configuring routes and attaching certificates, cookies, and tokens to HTTP requests, resulting in difficult-to-debug errors."*

Consistent with other studies applying qualitative methods to analyze Q&A posts [? ? ? ?], 50 posts were randomly selected from the 'top' 200 posts after sorting by number of views and score. Open coding [?] was applied for this analysis to guide the identification and synthesis of concepts related to the context, problem, and subject under test for each of the 50 posts. This was first performed independently by two authors, who then discussed and merged the code sets collaboratively to ensure agreement. Codes were sorted, grouped, and combined as needed until a set of distinct pain points emerged that captured all of the challenges observed. As a result of this analysis, seven pain point categories were identified; these are discussed in-depth in Section 6 and summarized in Table 3.

4.3 Analysis of Linked Resources

To supplement findings from our topic model analysis, which considered only the text content from the title and body of each post, we also examined resources linked by developers in both questions and answers. Developer Q&A forums allow users to include non-text content such as code snippets or links in their posts, answers, and comments. To understand the types of resources developers use and recommend for security unit and integration testing, we extracted links from all posts using the following process:

(1) **Collection:** During preprocessing (Section 4.1.1), 1,320 links were extracted from our post dataset: 461 links were shared in the body of a question and 859 were provided in answers.
(2) **Filtering:** Links were then manually filtered to remove those that were broken (e.g. 404), deep (e.g. localhost), or images. As a result 250 "irrelevant" links were excluded, leaving 1,070 relevant links: 304 from questions and 766 from answers.
(3) **Domain Extraction:** The Python library [?] was used to extract the domain and subdomain from each link. For example, the subdomain of http://guides.rubyonrails.org/testing.html is '' and its domain is ''.
(4) **Aggregation:** Next, aggregated frequencies were calculated for unique domains and subdomains: 128 were identified from question links and 290 from answer links.
(5) **Top Domains**: Unique domains were sorted by frequency to determine the top 5 most shared sites in each group.
(6) **Qualitative Analysis:** A set of randomly-selected links from each of the top domains were manually reviewed to identify the title and subject and investigate any patterns.

5 RQ1 RESULTS: DISCUSSION TOPICS

The 10 topics produced by the LDA model are shown in Table 2. Section 4.1.2 describes the tuning experiments conducted to identify the optimal model configuration (number of topics, text preprocessing values, *etc.*) for the dataset. The resulting topics (except *Error*) represent the security controls, components and scenarios that developers commonly test at the unit and integration levels. *Encryption* and *Authentication* were the most-discussed security controls; consistent with the complex and highly customizable nature of authentication [?], distinct topics emerged for three protocols: *Cookie*, *Token*, and *Basic.* Frequent discussion of concepts related to *Login* and *Protected Resource* indicate these are frequently-tested scenarios imply for authentication and authorization controls.

Developers frequently discussed *HTTP Requests*, *Client/Server* and *SSL Certificates* in their security testing posts, suggesting many seek help testing scenarios involve these components. Finally, the appearance of *Error* as a distinct topic is not surprising considering the data comes from Q&A websites but implies developers tend to seek help solving problems faced during test execution.

RQ1 Key Findings:

- Using an LDA topic model, **10 key discussion topics** were identified from 525 security unit and integration testing posts mined from Stack Overflow and Security Stack Exchange.

Table 2: Results for RQ 1: Key Discussion Topics

Topic Label	Terms
SSL Certificate	"certif", "ssl", "api", "applic", "error", "perform", "respons", "key", "server", "issu"
Encryption	"integr", "code", "unit", "encrypt", "write", "control", "call", "exampl", "method", "authent"
Cookie Authentication	"user", "app", "authent", "password", "cooki", "work", "load", "login", "web", "fail"
Token Authentication	"user", "applic", "api", "sign", "author", "token", "creat", "integr", "id", "work"
Basic Authentication	"password", "connect", "server", "usernam", "like", "would", "unit", "way", "basic", "client"
Login	"login", "page", "valid", "script", "work", "password", "credenti", "user", "usernam", "pass"
Protected Resource	"web", "authent", "access", "ad", "file", "like", "creat", "key", "want", "servic"
Client/Server	"jmeter", "unit", "certif", "client", "token", "server", "ad", "http", "send", "valid"
HTTP Request	"request", "servic", "secur", "applic", "log", "http", "authent", "code", "work", "would"
Error	"error", "develop", "work", "fail", "gener", "problem", "code", "server", "follow", "configur"

- The first **nine topics represent commonly-tested security controls, scenarios and components**
- The final topic (*Error*) reflected the nature of the discussion around the posts, *i.e.* debugging test failures and errors

6 RQ2 RESULTS: PAIN POINTS

The qualitative analysis revealed 7 pain points experienced by developers when writing security unit and integration tests, summarized in Table 3. They span all test phases and affect tests involving the nine security controls, scenarios, and components identified by the topic model in Section 5.

6.1 Mocking Security Components

Before individual test methods can be written, a test environment (*i.e.* fixture) must be setup to satisfy *preconditions* that recreate the system state needed for each test scenario. For example, an active session with an authenticated user must be established to test logout scenarios for a web application. Fixtures are often initialized in methods accessible by test methods to ensure the consistent system state needed for repeatable tests. As part of this process, it is often necessary to simulate (*i.e.* mock) some system components, objects, or functionality. For example, edit or delete scenarios may be tested with fake users to avoid compromising the confidentiality, integrity, or availability of real user's data. Mocking security-related components was the most frequently-observed pain point in our entire analysis. Attempts to write tests without mocking (when appropriate) or incorrectly implementing a mock often lead to errors and unexpected test failures that were difficult to debug. We did not investigate the rationale behind the mocking decisions observed, but a prior study by Spadini *et al.* [?] suggests developers are often influenced by application-specific factors and individual preferences. Correctly recreating interactions between mocked fixtures and "real" system components can also be a frustrating process. In the example given for this pain point in Table 3, the post author is trying to test OAuth-based authentication and authorization controls that were implemented with the Spring Security framework. Their test fixture uses the class provided by Spring Security to simulate communication with API endpoints. Despite referencing multiple tutorials and related Q&A posts, the author can not figure out why their custom authentication provider is not being called in this context, which causes the given test method to throw a 401 error when executed. None of the three answers to this question were accepted, but the discussions indicate that the author has missed a configuration step.

6.2 Interacting With Security Services & APIs

A wide variety of services and APIs are available that developers can use for their control implementations. This is considered a good practice [?] because it reduces the developer's responsibility for correctly implementing the actual functionality of complex controls. Instead, developers integrate them into their code bases and configure mechanisms to enable distributed communication. Many of the posts examined involved distributed third-party services for authentication like OAuth, Okta, and OpenSSO, and OpenID. Developers struggled to recreate these interactions within their test environments. They asked for help constructing tests for scenarios that require accessing and manipulating session properties, communicating with an application's web API, and communicating with servers within test environments. Interactions between services and APIs that are distributed (outside of the test environment) and/or controlled by a third-party were especially difficult. Other types of services were also being used in the posts examined; the example post provided for this pain point in Table 3 is asking how to unit test interactions with a Paypal (financial transactions) API. Based on the question description, the author was attempting to communicate with the API's live interface from a sandbox account, but did not know how to simulate buyer authentication. In this case, the accepted answer suggests to mock the interface instead, and links to documentation for the sandbox to help. PayPal's developer documentation site was the fourth most-linked resource in questions but was not a top answer in domains (Section 7), which indicates the documentation may lack appropriate guidelines.

6.3 Creating User Fixtures

Security controls are designed and built into software to protect the confidentiality, integrity, and availability (CIA) of system services and data. The these controls must therefore be tested to ensure they have been correctly implemented and data is not compromised when exercised under different conditions and scenarios. Naturally most security functionality involves manipulating data related to *users*, especially credentials. Creating user fixtures may be a painful experience when testing other components like databases that interact with user data. However, testing security control behaviors requires unique access to and manipulations of user data, namely credentials, that warrant its inclusion as a *security* unit and integration testing pain point. Namely, developers

Table 3: Results for RQ2: Summary of Security Unit & Integration Testing Pain Points

Pain Point	Description	Example (Excerpts from a Related Question)
Mocking Security Components	Developers struggled to correctly design and configure mocks for complex security components or providers implemented using third-party security frameworks (*e.g.* Spring Security) and had trouble debugging associated test failures and errors.	*"I'm **trying to test my spring OAuth2 authorization and authentication...using spring's MockMvc class**...The fundamental issue I'm facing is the fact that **my custom authentication provider is never called** even if I have registered it as one of the authentication providers used by spring security." (Post ID: 49079406)*
Interacting with Security Services & APIs	Not knowing "how to test" scenarios that involve distributed or third-party security services (*e.g.* OpenID) and APIs (*e.g.* Paypal) can be an obstacle, often due to uncertainty of what should be simulated.	*"I want to write **unit test to test my Paypal Express Checkout integration**. I have **problem in the step where buyer authorize payment** in Paypal screen...**Is there a way to simulate this action in my test code?"** (Post ID: 19763494)*
Creating User Fixtures	The unique ways security controls interact with user data make this task particularly frustrating when testing login and password validation scenarios. Developers struggle to understand how to design appropriate fixtures, and incorrectly configured credentials can lead to confusing test failures.	*"**I've verified that [fake user fixture] is correctly being loaded** into the test database time and time again. I can grab the User object...I can verify the password is correct...The user is active. **Yet, invariably, [login command] FAILS**. I'm baffled as to why. I think I've read every single Internet discussion pertaining to this." (Post ID: 2619102)*
Bypassing Access Controls for Protected Resources	The procedure for simulating authenticated state or "skipping" authentication often depends on the system-under-test's design and the security tools, libraries, etc. used to implement these controls. Developers sought help designing tests and debugging errors and failures caused by missing or incorrectly-implemented steps.	*"I currently have an ASP 5/ASP Core Web API that I need to **integration test with the OWIN Test Server**...I use IdentityServer as the authorization server in production and **I do not want to include the authorization as part of my integration testing**...How can I successfully **fake authenticate the user and bypass the [Authorize]**...?" (Post ID: 37223397)*
Designing Authentication Flows	Due to the complexity and framework-specific properties of most authentication control implementations, developers had a hard time identifying the sequence of events ("flow") needed to recreate low-level scenarios and mock components.	*"...**I want to login a user...using forms authentication**...My question is **how do I write a test to justify this code?** Is there a way to check that the method was called with the correct parameters? Is there any way of injecting a fake/mock FormsAuthentication? (Post ID: 366388)*
Creating HTTP Request Objects	Errors caused by incorrectly configured login routes and trouble attaching certificates, csrf tokens, or credentials to HTTP Requests were obstacles to testing scenarios involving security controllers and authentication. Usually developers knew how to do this in production but were confused about how to do so in their test environment.	*"...In writing **functional tests for [their API's user authentication controller]**...I am **running in to an issue testing HTTP Basic auth**... I have found numerous blogs that mention [code snippet] should be used to **spoof headers**...[authenticate method] does not see the headers and therefore is **returning false even in the presence of valid credentials**. (Post ID: 1165478)*
Configuring Systems for Test Environments	Identifying the appropriate system settings, build configurations, *etc.* that must be adjusted, and the correct approach to change them, was a painful process for developers who wanted to use custom or mocked components. Settings related to certificates and (real) distributed components were especially difficult.	*"I have **tried instructing Maven to use a local keystore**, the one that comes with the JRE, in an effort to **keep the expired cert** [used by Embedded Glassfish] **from being used**...the **expired cert is still being found** in whatever trusted keystore it defaults to." (Post ID: 18304232)*

struggled with user fixtures when testing login and password validation scenarios. There was a clear lack of awareness that the unique relationship between security controls and users should influence the design of these fixtures. Posts seeking design recommendations ("How do I test this?") indicate developers did not know how to create user objects, and discussed options like creating local user objects with hard-coded credentials and dynamically initializing fake users to store in a database. Developers also had a hard time debugging frustrating test failures caused by incorrectly configured user fixtures. Table 3 lists an example question for this pain point that demonstrates these challenges. The author cannot figure out why a login test for their Django application fails, describing their fixture setup and the code used to initialize users. The accepted answer points out that this problem can be traced to incorrect initialization of the user object. The question author incorrectly initialized their user due to an apparent misunderstanding of how the user creation method handles the credential parameters. accidentally set their raw password string as the hashed value. Because they did not realize this method interpreted the password argument as the hashed value, they used the same string to call the login function. The author allegedly looked for answers elsewhere (*"I think I've read every single Internet discussion pertaining to this."*) before posting, but does not disclose whether they checked documentation for the framework used to implement the login function.

6.4 Bypassing Access Controls

An important observation stemming from this pain point is that developers may need to write tests involving security controls even when they are not the subject-under-test. *Protected Resources* was identified as key topic in the full dataset, and in the manual analysis a clear pattern was observed in which developers sought help designing and debugging tests that attempt to bypass access controls. The appropriate procedure to achieve this often depends on the system-under-test's design and the security tools, libraries, etc. used in the implementation, and several posts sought help debugging errors and failures caused by missing or incorrectly-implemented steps. In the example post for this pain point (Table 3), the author asks how to write tests for their web API that bypass the authorization server used in production. The accepted answer is from the post author themselves, who seems to have identified the correct configurations needed to do so. These observations also show pain points can emerge even when developers want to avoid testing security-related scenarios.

6.5 Designing Authentication Flows

The fine-grained nature of unit and integration testing forces developers to recreate the scenarios-under-test using a similar process applied to implement them, *i.e.* executing a sequence of statements calling methods or manipulating isolated components. The complexity of security control implementations made it difficult for developers to identify correct sequences, or "flows" for many scenarios, but observations made during post analysis indicate developers had an especially hard time designing *authentication* flows. This may be influenced by the wide variety of protocols that can be used, as evidenced by the emergence of *Token, Cookie, & Basic Authentication* as distinct key discussion topics in the entire post dataset. The example post (Table 3) for this pain point follows a common ("how do I test this?") question pattern, in this case asking how to test form authentication (the protocol was not specified). The only (accepted) answer provides a code snippet explaining the technology-specific process, but neither this response or any of the comments link to any resource that would help. The need for more example-based [**?**] security documentation and usage guidelines [**?**] for testing at this level is discussed in Section 8.3.

6.6 Creating HTTP Request Objects

HTTP Requests are an essential aspect of distributed interactions which are a significant factor for web application development, including but not limited to *Client/Server* communication. Security functionality is often facilitated through such requests (*e.g.* submitting a login form) due to the distributed nature of security controls implementations. Even though request-based communication is a well-known concept that is not specific to security *or* testing, developers in the posts analyzed described challenges that made this task a uniquely painful experience when writing security unit and integration tests. They had the most trouble constructing requests for scenarios involving mocked security fixtures; when creating requests for authorization and authentication scenarios, developers asked how to attach *SSL Certificates*, credentials, and tokens. Other posts revealed difficulty or confusion when selecting the correct request headers and configuring routing behavior. In the example given for this pain point (Table 3), the post author asks for help debugging an unexpected failure they encountered when testing the *Basic Authentication* component of their web API. In the description, the author implies they have tried using recommended methods for spoofing request headers from "numerous blogs" (not linked) with no success. The only (accepted) answer provides a solution via code snippet but does not link other resources.

6.7 Configuring Systems for Test Environment

Aside from creating test fixtures, other system settings related to build and runtime behavior may need to be specially configured to correctly recreate the necessary state using a mixture of real and simulated components. In this context, developers asked questions about how to secure storage of sensitive data, change settings for system components they have limited access to, configure build tools to use mocked certificates, and set runtime to trust all certificates. Configuring settings related to *SSL Certificates* was difficult for developers whether or not they were using real certificates. In the example (Table 3) given for this pain point the post author is seeking help configuring their Maven build tool to reference a local keystore rather than the default used in production in an effort to circumvent an issue they had with their Embedded Glassfish testing server, which was using an expired certificate.

6.8 Common Factors Across Pain Points

Although each pain point identified is distinct they can co-occur, *e.g. Interacting with Security Services and APIs* and *Bypassing Access Controls for Protected Resources.* All pain points were influenced by the inherent complexity and customizability of security control designs. The third-party services, tools, libraries, and frameworks developers rely on to implement them abstract away much of the functionality, which can compound these negative influences. This is the driving force behind pain points such as *Designing Authentication Flows* but play some role in making all of the tasks in Table 3 painful.

RQ2 Key Findings:

- Qualitative analysis of 50 Q&A posts revealed **7 security unit and integration testing pain points** (Table 3).
- These pain points impeded efforts to write tests involving the security controls, components, and scenarios identified as key topics across all 525 posts mined for the study (Table 2).
- Pain points affect every test phase, but **test design, especially fixture setup, was most challenging.**

- All of the pain points identified were influenced by the inherent complexity of security control designs, which was compounded by the abstraction and language-specific challenges introduced by the third-party security tools and code.

7 RQ3 RESULTS: LINKED RESOURCES

An analysis of the top domains from resources linked by developers in our dataset of 525 security unit and integration testing Q&A posts are presented below. Links from questions and answers were analyzed separately to allow distinctions and patterns to be identified. We extracted a total of 476 links from our dataset: 160 unique question links and 316 unique answer links. The most frequently-shared resources in both the questions and the answers were links to other Stack Overflow posts. The following sections discuss the topics covered by a randomly selected set of links from the top domains of each group as shown in Tables 4 & 5.

7.1 Analysis of Resource Links from Questions

From the set of 38 stack overflow post links shared in questions, 19 were randomly-selected and manually reviewed to identify associations with the topics (Section 5) found by the LDA analysis.

Table 4: Resources Linked in Questions

Resource	Frequency
stackoverflow.com	38
github.com	36
gist.github.com	7
developer.paypal.com	6
w3.org	6

Nine posts were related to authentication, and discussed using the Passport.js library in tests, authenticating to PayPal sandbox, and *Basic Authentication*. Three links discussed mocking and testing *SSL Certificates*, two asked questions about tests involving cookies for Ruby On Rails applications, and two discussed the Java performance testing tool JMeter. Other topics mentioned in post links were Encryption, Error, and HTTP Request once each. Overall, 8 topics from Section 5 were discussed in the 19 Stack Overflow links.

GitHub repositories and files were the second most-shared resource in the questions. Nine randomly selected GitHub links were reviewed; these repositories were for tools, libraries *etc.* related to SSL Servers for Django (django-sslserver[**?**]), cryptographic algorithms (bouncy-castle-pgp[**?**]), Android user scenario testing (Robotium[**?**]), Android security (android-FingerprintDialog[**?**]), certificates (mkcert [**?**]), Authorization for Laravel (authority-laravel[**?**]), and three deprecated repositories. Three of the GitHub *gists* (snippets) were analyzed; their content was related to authentication testing on Symfony 1.4, a Karma config file, and authentication testing with JavaScript. From developer.paypal.com four resources were randomly analyzed; they were related to PayPal sandbox, Paypal REST SDKs providing boilerplate integration code in several programming languages, Paypal REST APIs relate to *Token Authentication*, and the developer documentation homepage. The w3.org links were all related to XML.

7.2 Analysis of Resource Links from Answers

Table 5: Resources Linked in Answers

Resource	Frequency
stackoverflow.com	68
jmeter.apache.org	63
github.com	60
blazemeter.com	37
en.wikipedia.org	27

The most shared resource for answers were other Stack Overflow posts. Qualitative analysis of fourteen randomly selected SO links revealed the following: six links discuss authentication for Ruby on Rails, PayPal, and *Basic Authentication*, which is very similar to the topics in the SO posts linked in the questions; three posts discussed using SSL certificates; another three talk about PayPal authentication and using the sandbox service; two focused on *HTTP Requests* and another two on *Encryption. Cookie Authentication*, JMeter, *Token Authentication*, were each mentioned once.

The second most shared resources in the answers was the home page to the JMeter performance testing tool; this tool and Blazemeter (the fourth top answer resource) are both related to the *Client / Server* topic. As shown in Table 2, JMeter was a highly infuential term for this topic. The JMeter links all pointed to the tool's documentation, while the Blazemeter links led to blogs discussing *HTTP Requests*, and *Cookie Authentication*. In the answers, GitHub was the third most-shared resource, and thirteen of these links were analyzed: three led to specific files and the other ten to tool, library, *etc.* repositories. Eight were related to authentication, two to *Encryption*, and the others discussed general topics like 'Apple Pay integration' and 'secure configuration of Ruby-on-Rails applications'. Finally the fifth most shared resource was Wikipedia and 10 randomly selected links were analysed: four discussed *Encryption*, two relate to authentication, and the following were each mentioned in one link: *SSL Certificates*, *HTTP Requests*, fuzz testing, and 'htaccess'.

RQ3 Key Findings:
- **Stack Overflow posts were the most commonly shared resource** in questions *and* answers.
- The other top resources represented **documentation and repositories** for security and testing tools, libraries *etc.*
- These findings **reflect the difficulty developers had with tests involving third-party code and tools** (*e.g.* JMeter)
- The majority of linked resources are related to Authentication, but 8 of the 9 commonly-tested security controls identified by the topic model (Table 2) were discussed.

8 DISCUSSION

8.1 Implications for Shift-Left Security

This work was motivated by the disconnect between efforts in the practitioner and research communities to develop tools and strategies for shifting security left into development workflows and the lack of empirical data on developer's current experiences writing automated tests involving security controls that should influence these efforts. The ubiquity of unit and integration testing means software projects usually already have the necessary infrastructure and writing these tests is a familiar practice for most developers. Therefore, shifting security testing to this level can be achieved by helping developers adapt their existing skills to this context instead of introducing an entirely new tool. It follows that the identified pain points may be barriers to success and should therefore influence the development of strategies and guidelines.

The importance of taking such a data-driven approach to shifting security left was demonstrated in this work by the identification of 7 pain points *unique* to security-related tests. Further, generic tasks like *Creating HTTP Request Objects* and *Creating User Fixtures* can be unexpectedly difficult due to the unique properties of security control implementations. Each of the pain points was closely associated with one or more of the nine security controls, components, and scenarios that were revealed as key discussion topics amongst the 525 Q&A posts mined for this study. Conducting and synthesizing both analyses strengthens their independent findings and supports the conclusion that these pain points represent *common challenges* affecting *commonly tested security controls*.

8.2 Implications For Practitioners

Despite security's classification as a "non-functional" requirement and lingering perceptions of it being a post-development concern, security controls are ultimately realized within system implementation. This holds even when external third-party tools or services are used to add security functionality, because the system under test will still need some mechanism to facilitate that communication. Given the ubiquity of security controls it is reasonable to assert that **most developers will likely need to write security-related tests and may experience these challenges *even if security testing is not the explicit motivation*** so these findings are also applicable to the development community at-large.

The first step towards mitigating these unique and potentially unexpected challenges is *awareness*. Many of the pain points begin or can be traced back to test fixture/environment design, where developer's uncertainty or incorrect execution of the appropriate process to configure security-related fixtures. These findings suggest that **test cases that will evaluate or include security-related code should be explicitly identified during test planning**. This activity would force a team or developer to *actively* and *preemptively* think about which security-related classes, components will be involved, how these should be realized in the test environments, and *reveal points of uncertainty or potential complexity*. This could be extended to create resources for long-term support that can be shared amongst a team, such as reusable security fixtures classes.

8.3 Research Directions

This study was a first step towards identifying potential barriers to successfully shifting security left into development workflows. These findings should be applied to the development of strategies and guidelines focused on addressing these challenges. Further, researchers who may be interested in improving testing experiences outside of the security context should recognize that these pain points affect developers *now*, even if security testing is not explicitly or apparently their driving motivation. The following are interesting research directions inspired by these results:

Extensions 525 Q&A posts related to security unit and integration testing from Stack Overflow and Security Stack Exchange were mined and analyzed in this study to learn pain points developers experience and examine associations between these challenges and commonly tested security controls, scenarios, *etc.*. Additional empirical studies should be conducted to extend and enrich understanding of security unit and integration testing challenges.

Guidelines for Security Unit and Integration Testing Challenges inherent to developer testing are well studied, and many resources enumerating best practices, patterns, *etc.* for avoiding them are available [? ?]. Likewise, there are guidelines and best practices for security testing such as those provided by OWASP [? ?]. Security unit and integration testing lies at the intersection of these concepts, but presents unique challenges and developers will need new guidelines to address them. This observation is consistent with findings from Acar *et al.* [?]'s survey of security resources for developers: testing guides focused on post-deployment techniques and support for 'automated testing' tools was restricted to static analyzers or fuzzers. The top resource linked in the posts reviewed for this study (Section 7) was Stack Overflow, and other top resources were repositories and documentation for security tools, APIs, and libraries. Nevertheless, none of the guidelines reviewed in [?] would help developers *use* security libraries/APIs, and Nadi *et al.*'s study of challenges with Java cryptography APIs suggests that documentation provides insufficient examples of API *usage* [?].

Automated Generation of Security Fixtures: Unit and integration testing research is mostly centered on removing the responsibility from developers though the automatic generation of test cases, oracles, and inputs. However, the findings from this study indicate developers would benefit from automated tools to *assist* with test design. Research focused on these problems could examine potential strategies for automatic generation of security fixtures (especially mocks), which was a major pain point observed in this study. Some language-based automated solutions have been proposed (*e.g.* [?]), but the unique complexity and abstraction of security controls make them a prime candidate for focused efforts.

9 THREATS TO VALIDITY

This section acknowledges threats to the validity of our findings associated with the study design and explains steps taken to address them. Regarding external validity, we do not claim that the set of topics and pain points derived from 525 and 50 pots respectively are exhaustive, but findings from qualitative and quantitative analyses were compared and considered holistically to strengthen confidence in their representation. Further, domain or technology dependent factors were not used to define pain points to ensure generalizability. The limitations of keyword-based methods of identifying posts related to a specific subject [?] can affect internal validity (*i.e.* capturing all available data), which was mitigated by sourcing candidate posts from a security-focused site (SSE) and basing the SO search on a pre-curated dataset of security-related post IDs [?]. The keyword-based filtering that was used to ensure relevant to unit and integration testing specifically was manually evaluated for recall and precision. Construct validity threats were mitigated using techniques from prior work [? ? ? ?], *i.e.* randomly selecting 50 of the top 200 most-viewed posts.

10 CONCLUSION

This study aimed to investigate potential barriers to shift-left security testing at the unit and integration levels using 525 Q&A posts from Stack Overflow and Security Stack Exchange. A mixed-method empirical study was designed which used unsupervised learning (LDA) to identify nine security controls, scenarios, and components frequently tested by developers, supplemented by qualitative analysis of 50 to identify 7 unique pain points associated with testing these subjects. This was enriched with an analysis of embedded links to map the topics discussed to the most commonly-shared resources. Based on these findings, broader impacts were discussed, practitioner recommendations were provided and new research directions were highlighted.

ACKNOWLEDGMENTS

This work was supported in part by awards CNS-1816845, CCF-1943300 from the National Science Foundation.