An empirical study of developers' discussions about security challenges of different programming languages

Roland Croft[1,2] · Yongzheng Xie[1] · Mansooreh Zahedi[3] · M. Ali Babar[1,2] · Christoph Treude[3]

Abstract

Given programming languages can provide different types and levels of security support, it is critically important to consider security aspects while selecting programming languages for developing software systems. Inadequate consideration of security in the choice of a programming language may lead to potential ramifications for secure development. Whilst theoretical analysis of the supposed security properties of different programming languages has been conducted, there has been relatively little effort to empirically explore the actual security challenges experienced by developers. We have performed a large-scale study of the security challenges of 15 programming languages by quantitatively and qualitatively analysing the developers' discussions from Stack Overflow and GitHub. By leveraging topic modelling, we have derived a taxonomy of 18 major security challenges for 6 topic categories. We have also conducted comparative analysis to understand how the identified challenges vary regarding the different programming languages and data sources. Our findings suggest that the challenges and their characteristics differ substantially for different programming languages and data sources, i.e., Stack Overflow and GitHub. The findings provide evidence-based insights and understanding of security challenges related to different programming languages to software professionals (i.e., practitioners or researchers). The reported taxonomy of security challenges can assist both practitioners and researchers in better understanding and traversing the secure development landscape. This study highlights the importance of the choice of technology, e.g., programming language, in secure software engineering. Hence, the findings are expected to motivate practitioners to consider the potential impact of the choice of programming languages on software security.

Keywords Software security · Repository mining · Natural language processing · Empirical software engineering

1 Introduction

A recent successful cybersecurity attack on FireEye,[1] one of the World's largest security companies, is yet another indicator of the rapidly increasing security threats to digital systems (Rafter 2019), which are enabled by software. To ensure the availability, confidentiality and integrity of a software system, it is vital that organisations resolve software security issues that may be introduced during development and/or evolution of a software system (Shahriar and Zulkernine 2012). That is why there are increasing calls for putting more emphasis on software security issues throughout software development and evolution by incorporating the principles, practices, and tools of software security (Mailloux and Grimaila 2018). There is an important need to provide developers with valuable security information sources and support to achieve secure software development; most novice developers lack the required security expertise (Barnum and McGraw 2005).

Whilst developers play a key role in the choice of an implementation technology, e.g., programming language, for a software development project, developers may tend to consider the choice of a programming language from the perspectives of project requirements, personal experience, learning costs, language popularity, and the availability of toolkits/packages (Meyerovich and Rabkin 2013). Given that development security flaws and weaknesses can cause major software security problems, it is important to consider the security issues when choosing programming languages, which are known to provide different types and levels of support for security (Shahriar and Zulkernine 2012). For example, the C or C++ programming languages can have un-trapped memory management errors that make them unsafe; however, the Java programming language provides static and dynamic checks that ensure no un-trapped memory management errors can occur. We assert that it is important to increase awareness and knowledge about the security issues encountered while using different programming languages. Our research, reported in this paper, purports to empirically identify the security challenges that developers experience while using different programming languages. We decided to analyse and synthesize software developers' observations and experiences of security issues related to different programming languages shared on Open Data platforms like Stack Overflow and GitHub.

To achieve this objective, we have conducted a large-scale study of developers' discussions about security issues related to popular programming languages using quantitative and qualitative methods. For this study, we examine over 20 million issues from 27,312 GitHub repositories and over 11 million posts from Stack Overflow, for 50 of the most popular programming languages. We have then conducted an in-depth analysis of 136,400 issues from 7194 GitHub repositories and 143,498 Stack Overflow posts for 15 of the most popular programming languages such as Java, C, Python, and PHP.

We assert that this study has gathered and analysed the largest amount of security issue related data generated from developers' discussions on Stack Overflow and GitHub. This study also provides useful insights into three major software engineering aspects: the projects, the problems and the users. Whilst a few efforts have performed exploratory analysis of the security topics discussed by developers on Stack Overflow (Yang et al. 2016) and GitHub (Zahedi et al. 2018), there has been no effort aimed at empirically exploring the security issues from a technology or programming language perspective. Furthermore, we are the first to conduct comparative analysis of the experienced security challenges across data sources and domains. Previous studies have similarly investigated the characteristics

---

[1] https://www.nytimes.com/2020/12/08/technology/fireeye-hacked-russians.html

of Stack Overflow security discussions (Bayati and Heidary 2016; Lopez et al. 2018; Lopez et al. 2019), to better understand the nature and ways in which security knowledge is shared. We extend this analysis by comparing the characteristics of specific security topics and domains to investigate the explicit differences in the challenges by examining the discussion popularity, difficulty, and expertise. The main findings from this study are:

– Overall, the security challenges are increasing for most languages, except for PHP. Newer languages (i.e., Swift and TypeScript) experience a constant rise in security challenges after release.
– Security challenges and trends often cluster based on the tasks that the languages are commonly used for. GitHub security issues of Web-oriented languages (e.g., JavaScript, TypeScript and PHP) receive the most popularity, and GitHub users for Mobile-oriented languages (e.g., Java, C#, Objective-C and Swift) have the highest level of security expertise.
– Shell-based and Web-oriented languages experience significantly higher average rates of security discussion. Julia, MATLAB and R (languages primarily for scientific programming) exhibit exceedingly small numbers of security discussion.
– C/C++ are the only languages which prominently face challenges for memory management, and their questions on Stack Overflow receive some of the most popularity and attention from expert users. C/C++ users also have the most consideration for security via the intent of their discussions.
– There is a disconnect in the nature of the language security challenges between Stack Overflow and GitHub. In particular, Web-Development languages and Authentication challenges are considered easier and less popular on Stack Overflow than they are on GitHub.

The findings from this study are expected to provide better understanding and useful insights about the security issues experienced when using different programming languages. Software development practitioners can gain better understanding of the influence that programming languages have on software security. Developers can be motivated to consider the security issues associated with different programming languages when choosing and using different programming languages in software development projects. The main contributions of our work reported in this paper are as follows:

– It carries out a first of its kind empirical study of security issues for different programming languages by quantitatively and qualitatively analysing the largest amount of data of developers' discussions on Stack Overflow and GitHub. The findings contribute to the evidence-based body of knowledge about the security issues related to programming languages.
– It identifies a taxonomy of 18 security topics and 6 security categories that are commonly discussed and faced by developers during software development. The identified taxonomy can help categorize known security issues related to different programming languages as a source of guidance for developers.
– It provides a method that enabled us to extract, cluster, and analyse a large number of domain specific discussions from both Stack Overflow and GitHub. The reported method helped us to construct and make publicly available the largest, to our knowledge, dataset of developers' discussions about security issues of 15 popular programming languages.[2]

---

[2]https://github.com/RolandCroft/Language-Security-Challenges/

The rest of the paper is organized as follows. In Section 2, we discuss the background and motivation of this study. In Section 3, we present the methodological and logistical details for conducting this research. We report and analyse the results in Section 4. We discuss the results and highlight the implications of this research for practitioners and researchers in Section 5. The threats to validity and their implications are presented in Section 6. Section 7 draws some conclusions and indicates the possible future work.

2 Background and Motivations

2.1 Programming Language Software Quality Analysis

Programming language paradigms define the characteristics of a programming language and provide classification (Pierce and Benjamin 2002). In Table 1, we categorize 15 popular programming languages using a classification system inline with previous works (Ray et al. 2014; Zhang et al. 2019). We note that there are no commonly agreed definitions for language types. It is also known that paradigms are often relative rather than absolute. We define the type safety as whether a language prevents operations between mismatched types (strong) or allows type inconsistency (weak) (Cardelli and Wegner 1985). Type checking determines whether type errors are reported based on the source code (static) or the runtime behaviour (dynamic) (Cardelli and Wegner 1985). Memory management refers to the extent at which the programs handle memory safety, control and management; whether the responsibility is on the user (manual) or not (managed) (Dhurjati et al. 2003).

Many studies have investigated various attributes and aspects of programming languages (Horschig et al. 2018; Bhattacharya and Neamtiu 2011; Kleinschmager et al. 2012; Sestoft 2005; Hanenberg et al. 2014), through empirical analysis of historical software artefacts. However, they fail to provide a developers' perspective of the actual experienced challenges.

Only a few works aim to form a large-scale comparative analysis between programming languages. Ray et al. (2014) investigated the impacts of programming language paradigms on various code quality metrics, Kochhar et al. (2016) similarly investigated the effects of multiple programming languages on code quality, and Zhang et al. (2019) analysed the impacts of programming language on bug resolution characteristics. We aim to build upon the previous works and provide a security perspective.

Similar to how previous works indicate that choice of a programming language may impact code quality (Ray et al. 2014; Kochhar et al. 2016; Zhang et al. 2019), we extrapolate that programming language selection will also impact code security. For instance, type safe languages (languages which check for type errors (Cardelli and Wegner 1985)) are usually considered resistant to errors such as Buffer Overflow, as they handle typing and other memory management tasks automatically (Grossman et al. 2005). In contrast, a developer is responsible for ensuring safety and protection in weak typed languages, which leaves them more open to possible vulnerabilities and exploits. Thus, it is expected that the programming paradigms would impact the relative security and safety of different programming languages.

Khwaja et al. (2020) identify the different security vulnerabilities present in different programming languages, and the languages' support for prevention. Similarly, Cifuentes and Bierman (2019) perform a theoretical analysis of different languages capabilities to handle different vulnerabilities. However, they only consider a subset of vulnerabilities and their analysis remains purely theoretical; security flaws do not directly equate to development challenges.

Table 1 Categories of 15 popular programming languages

Category	Language	Java	PHP	JavaScript	C#	Python	Shell	C/C++	Ruby	Objective-C	Swift	PowerShell	Perl	Go	TypeScript
Type Safety	Strong	✓		✓	✓	✓			✓		✓	✓	✓	✓	✓
	Weak		✓				✓			✓					
Type Checking	Static	✓			✓			✓		✓	✓			✓	✓
	Dynamic		✓	✓		✓	✓		✓			✓	✓		
Memory Management	Managed	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓
	Manual							✓		✓					
Creation Date		1995	1995	1995	2000	1990	1989	1972	1995	1984	2014	2006	1987	2009	2012

Furthermore, the security requirements and challenges of software are dependent on the purposes it is used for (Sindre and Opdahl 2005). We would expect that network security and access control are especially important for web development, whereas resource management errors are more important for systems programming. However, different programming languages are suited better to different development tasks and domains. We outline some common development areas for 15 popular programming languages in Table 2, based on the common uses and professions of each language described in popular online blogs.[3] [4] Hence, we anticipate that the relative security requirements and considerations differ for each programming language based on the activities it is used for.

However, the extent to which these security requirements and challenges differ for each programming language is presently unknown, and current knowledge is based on theory, personal opinion, or anecdotal evidence. Hence, we aim to investigate developer knowledge, views and experiences to help build an evidence-based body of knowledge about programming language related aspects of secure software engineering.

2.2 Publicly Available Security Discussions

There exists an abundance of developer discussion and knowledge sharing in publicly available repositories. Question & Answering (Q&A) sites contain crowd-sourced knowledge on a variety of software development related tasks and subjects. Repository hosting sites document software development practices and issues. Thus, software researchers have started mining these data sources with the aim of understanding software practices and improving development processes.

Thus, we also utilize these two sources to obtain a snapshot of security-related developer activities and knowledge. Due to the nature of these discussions, either asking for help or identifying an issue, we refer to the topic of a discussion as a *challenge*. By obtaining knowledge of the security challenges that practitioners individually face for development in particular programming languages, we can form our comparative analysis of the manifested challenges to identify differences in their nature and characteristics.

Stack Overflow is a Q&A website for software development. We select Stack Overflow for our data analysis as it is the most widely used source for Software Engineering (SE) based knowledge sharing. As of December 2020, there have been over 20 million questions and 30 million answers posted by over 13 million developers since the websites inception in 2008. Each Stack Overflow post contains a unique id, owner, title, question post, tag(s) and answer post(s) along with additional meta-information such as comments, views, favorites and score. For the purposes of this study, we henceforth use the term *post* to refer to a question and all its respective answers. Example Stack Overflow posts are referenced in this paper via *(SO, XXX)* which can be accessed through a web browser via https://stackoverflow.com/questions/XXX. Figure 1 shows an example Stack Overflow question about preventing the security vulnerability SQL injection.

Stack Overflow is becoming of increasing focus to software researchers due to its diverse and thorough body of knowledge. In a survey conducted by Ahmad et al. (2017) they discovered over 500 papers related to mining Stack Overflow data. Le et al. (2020) construct a tool named PUMiner with the explicit purpose of mining security related documents from

---

[3]https://raygun.com/blog/programming-languages/

[4]https://www.computerscience.org/resources/computer-programming-languages/

Table 2 Common development domains of 15 popular programming languages

Development \ Language	Java	PHP	JavaScript	C#	Python	Shell	C/C++	Ruby	Objective-C	Swift	PowerShell	Perl	Go	TypeScript
Front-End Web			✓											✓
Back-End Web	✓	✓	✓	✓	✓			✓				✓	✓	✓
Mobile	✓		✓	✓					✓	✓				✓
Game	✓			✓			✓							
Application	✓			✓	✓		✓		✓	✓				
Systems							✓						✓	
Data Science					✓		✓							
Cloud	✓	✓	✓	✓	✓			✓					✓	
Command-Line	✓					✓					✓	✓	✓	

How can I prevent SQL injection in PHP?

Asked 11 years, 9 months ago   Active 3 days ago   Viewed 1.7m times

2773

🔖
3651

🔒 **This question's answers are a community effort**. Edit existing answers to improve this post. It is not currently accepting new answers or interactions.

If user input is inserted without modification into an SQL query, then the application becomes vulnerable to SQL injection, like in the following example:

```
$unsafe_variable = $_POST['user_input'];

mysql_query("INSERT INTO `table` (`column`) VALUES ('$unsafe_variable')");
```

That's because the user can input something like `value'); DROP TABLE table;--` , and the query becomes:

```
INSERT INTO `table` (`column`) VALUES('value'); DROP TABLE table;--')
```

What can be done to prevent this from happening?

php   mysql   sql   security   sql-injection

share follow   edited Oct 1 '16 at 8:08   community wiki
                                                                44 revs, 36 users 14%
                                                                Andrew G. Johnson

comments disabled on deleted / locked posts / reviews

28 Answers   Active | Oldest | **Votes**

Fig. 1 A Stack Overflow post about SQL injection prevention

Q&A sites like Stack Overflow. Whilst we do not explicitly adopt their tool, we derive heavy inspiration for the method of our data collection.

GitHub is an open-source platform for software repository hosting. We select GitHub as it is the most widely used public software development platform. As of December 2020, there are over 58 million registered users,[5] and over 41 million publicly hosted repositories.[6] Developers can stimulate discussion with other collaborators about ideas, enhancements, tasks or bugs via GitHub issues. We focus on GitHub issues to observe and analyse the problems and discussions that developers conduct during software development. An example GitHub issue discussing the possibility of an SQL Injection vulnerability in the software can be seen in Fig. 2. GitHub repositories also provide metadata about their programming language composition, so we associate discussions with the main programming language of that repository. We use the term *issue* to refer to a GitHub issue and all its associated comments. In this study, we reference example GitHub issues as *(GH,XXX)* which can be accessed through a web browser via https://github.com/XXX.
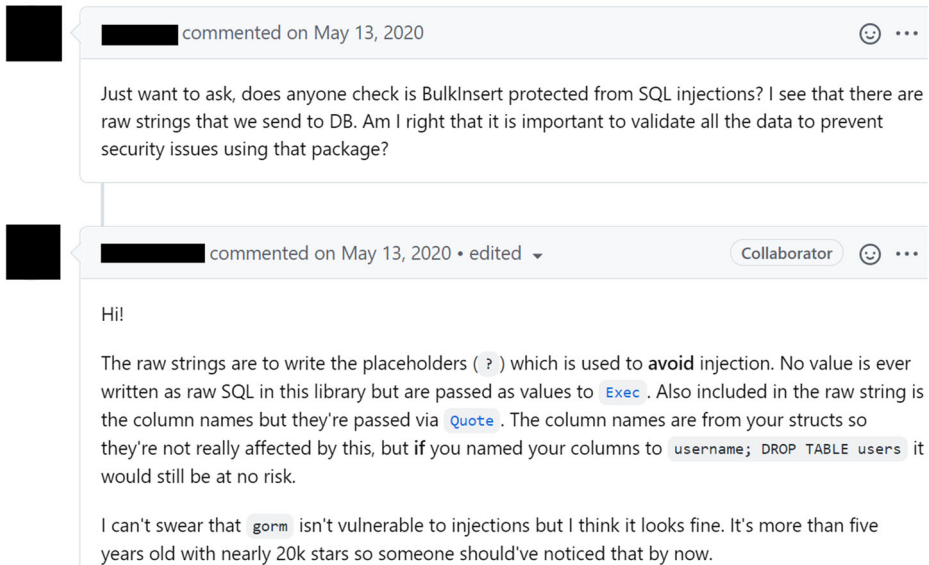
---

[5]https://github.com/search?q=type:user&type=Users
[6]https://github.com/search?q=is:public

[Question] SQL injection potential flaw? #38

Fig. 2 A GitHub issue about potential SQL injection

GitHub contains a much larger variety of data, and consequently its research applications are much more diverse. Kalliamvakou et al. (2014) conduct an empirical analysis of GitHub data and warn researchers of perils of mining GitHub for research purposes. In particular, they find that the majority of repositories on GitHub are personal, inactive or not related to software development. We attempt to avoid these perils by only considering the most starred repositories. Pletea et al. (2014) present sentiment analysis of security related discussions on GitHub. We build upon their method of data collection to extract security discussions from GitHub.

There are also a few studies which aim to draw connections between Stack Overflow and GitHub data. Vasilescu et al. (2013) investigate the activity correlation of users in Stack Overflow discussions and committers in GitHub repositories. They find that active GitHub users tend to ask fewer questions and provide more answers on Stack Overflow. A similar study is conducted by Xiong et al. (2017) who create an algorithm for linking developer identity on Stack Overflow and GitHub, and analyse their behaviour on both sites.

We observe Stack Overflow and GitHub to be two of the most prominent online sources for developer community support, due to their size and popularity. Hence, to obtain a more complete view of development security challenges, we intend to pull related insights from the discussions of both Stack Overflow and GitHub.

2.3 Topic Modelling of Software Engineering

Topic modelling is a statistical Natural Language Processing (NLP) technique used to auto-matically discover the topics of text corpora (Blei et al. 2003). Latent Dirichlet Allocation (LDA) is the most widely used implementation of topic modelling (Chen et al. 2016), and hence we also adopt it as the standard. LDA uses word frequencies and co-occurrence fre-quencies in the input documents to build probabilistic word and document models that uncover the hidden semantic structures of the text (Blei et al. 2003).

LDA has become a popular and effective tool for analysing large sets of software engineering related text data. Like us, many previous researchers have used LDA to automatically identify and cluster the topics of publicly accessible discussions on Stack Overflow for a variety of domains and aspects. Barua et al. (2014), Zou et al. (2015), and Allamanis and Sutton (2013) have all leveraged LDA to identify general topics and trends in Stack Overflow data. This type of research has also been targeted towards a variety of different domains: mobile (Linares-Vásquez et al. 2013; Rosen and Shihab 2016), web development (Bajaj et al. 2014), concurrency (Ahmed and Bagherzadeh 2018), big data (Bagherzadeh and Khatchadourian 2019), API usage (Campbell et al. 2013), machine learn-ing (Bangash et al. 2019; Han et al. 2020), blockchain (Wan et al. 2019), and security (Yang et al. 2016). Some studies have also aimed to analyse the discussions of GitHub repositories like we do. Rahman and Roy (2014) apply LDA to repository pull requests in order to help identify factors that contribute towards successful or unsuccessful pull requests. Zahedi et al. (2018) apply LDA to issue discussions in GitHub repositories to identify the main security issues that developers face. Due to its successful application to similar research domains as ours, we also utilize LDA to cluster the discussions and identify the main challenges.

Whilst some of our topics overlap with the existing security-related LDA research (Yang et al. 2016; Zahedi et al. 2018; Le et al. 2021), our overall taxonomy differs substan-tially as we avoid technology specific topics and ensure generalizability for comparison. Furthermore, due to the consideration of programming language our analyzed posts differ substantially. Reproducing the method for data extraction from previous works, we find our Stack Overflow dataset to be 1.5 times larger than the dataset obtained by Yang et al. (2016), and only 0.4% of our sampled repositories overlap with Zahedi et al. (2018). Additionally, our analysis of the challenges is more thorough as we also consider the popularity, diffi-culty and expertise. Finally, we perform extensive comparative analysis of the topics across programming languages and data sources.

3 Methodology

3.1 Research Questions

This study is designed to answer six major Research Questions (RQs) in order to provide a thorough analysis and understanding of security challenges for different programming languages.

– **Programming Language Security Consideration.** We first aim to provide a gen-eral indication of how comparatively important security is for different languages. We examine the overall amount of discussion for all languages to help select the most popular programming languages for analysis in subsequent research questions.

RQ1.   What is the rate of security discussion amongst programming languages on Stack Overflow and GitHub?
   This first research question aims to give a broad view of the security consideration for a large number of languages, via the rate at which security-related discussions occur.

– **Programming Language Security Discussions.** We answer three research questions related to the content of security discussions. Specifically, we aim to identify what security challenges are faced by developers for different programming languages and why.

RQ2.   What is the intention behind security discussions for different programming languages?
   This RQ seeks to identify why developers are discussing security topics for particular languages. We analyse the intent, motivations and purpose of the security questions and issues being posted on Stack Overflow and GitHub. Through this knowledge we can better determine why security is discussed differently, and the potential security information needs of developers.

RQ3.   What are the major security challenges and topics discussed by developers for different programming languages?
   Through RQ3 we aim to identify and compare the major categories of security challenges that developers face when using different programming languages. We also analyse the relative significance of these categories through their prevalence in developer discussions. Through this synthesis of publicly available knowledge, we enable practitioners to better understand the security issues commonly encountered for different languages, and provides researchers with a taxonomy of key security topics.

RQ4.   How do security discussion topics change over time for different programming languages?
   RQ4 aims to further this analysis by examining how the topics evolve over time, which provides insights into whether certain challenges become more or less important for specific languages as they update. RQ4 enables both practitioners and researchers to observe the relative trends and importance of the different security topics for different languages.

– **Programming Language Security Support.** Finally, we answer two research questions relating to the community support available for security challenges of different programming languages. We aim to identify the characteristics of how security related challenges are received and handled by the community. To achieve this, we use quantitative metrics derived from discussion metadata for popularity, difficulty and expertise, inspired from previous works (Yang et al. 2016; Ahmed and Bagherzadeh 2018; Tian et al. 2019).

RQ5.   What are the characteristics in terms of popularity and difficulty of different programming languages and their identified security challenges?
   RQ5 uses document-centric metrics to identify the relative popularity (engagement and interest) and difficulty (complexity and labour). The combination of these metrics gives an indication of how significant the security challenges are when they occur.

RQ6. What is the level of security expertise of the users who answer security related discussions for different programming languages?

RQ6 utilizes user-centric metrics to examine the relative expertise (knowledge and skill) of users of different programming languages. These characteristics help garner further insights into how the challenges are handled, i.e., is there sufficient knowledge and enough experts to help resolve security challenges. As such, we only consider the users who provide answers to the security related discussions.

3.2 Data Collection

To conduct our study we collected a dataset of security and programming language related discussions from Stack Overflow and GitHub. The workflow for the data extraction process is shown in Fig. 3. Our data collection process follows three major steps. First, we identified the programming languages commonly used by developers so that we could scope our study. Secondly, we extracted a dataset of discussions relating to each programming language from both GitHub and Stack Overflow. Thirdly, we filtered the datasets from step 2 to security-related discussions. Step 2 and 3 are performed separately so that we can identify the rate of security discussion as well as more general programming language discussion characteristics.

To first identify the programming languages used by developers, we selected the 50 most commonly used programming languages on GitHub in 2020 for consideration, as reported by *GitHut*.[7] We then obtained a programming language specific dataset for each of these 50 programming languages from Stack Overflow and GitHub.

For Stack Overflow, we extracted programming language specific data based on the tags of each post, similar to prior works (Yang et al. 2016). For the 50 identified programming languages, we manually identified the appropriate Stack Overflow tag, as well as all tags relating to different versions and releases of that language. The complete tag list can be viewed in our online appendix.[8] Every post on Stack Overflow that contains one or more of these tags is extracted; considering all data up to May 1st 2020.

If a post contains more than one language tag, it is included in each of the relevant language datasets. However, C and C++ are both syntactically and semantically similar, and have heavy overlap in their secure coding properties (Seacord 2005). Hence, this approach causes redundancy in the individual datasets of C or C++. The C dataset contain 2641 posts in comparison to the C++ dataset that contains 3136 posts. However, 323 of the 2641 C posts (12%) are also contained in the C++ Stack Overflow dataset. Furthermore, repositories primarily written in C often also contain C++ code. Hence, to reduce the redundancy in our analysis, we consider the dataset collected for C to sufficiently represent both the C and C++ language, due to their natural overlap and similar semantic properties. This redundancy was later confirmed in our results. When extracting topics for the C and C++ datasets individually, the six topic categories that manifest for C Stack Overflow posts are also the 6 most dominant topic categories for C++ Stack Overflow posts.

For GitHub, we extracted language specific data based on the predominant language of the repository. However, due to the substantially larger size of GitHub data, we were unable to efficiently analyze all repositories. Hence, we decided to use a repository sampling

---

[7]https://madnight.github.io/githut
[8]https://github.com/RolandCroft/Language-Security-Challenges/tree/master/Appendix/SOLangsTop50.txt
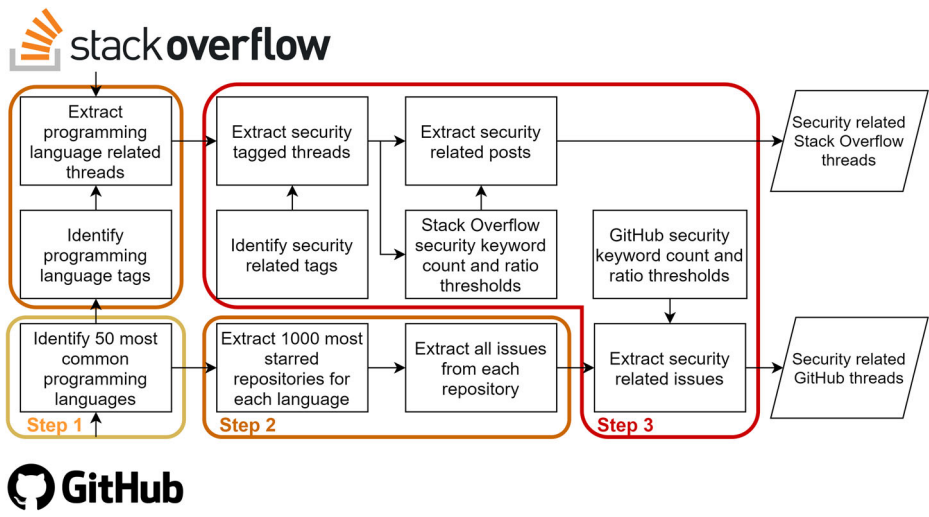
Fig. 3 Workflow of retrieving security related posts and issues from Stack Overflow and GitHub

method similar to the previous work by Zahedi et al. (2018). We retrieved the top 1000 most starred repositories for each programming language. To ensure that we only sampled high quality repositories for large scale software projects, only repositories with more than 100 stars are kept from the top 1000. For GitHub we extended the data collection time span slightly to increase the amount of available data for each repository, considering data up to July 1st 2020.

Thirdly, we filtered the programming language datasets to security-related discussions. As we aim to identify the security challenges that are discussed, it is important that the discussions in our dataset are relevant and focused towards security. To achieve this, we utilized both tag-based filtering and content-based filtering on our datasets; where applicable.

Tag-based filtering uses the tags of a post to identify the category of its content. This approach is most commonly adopted by previous works (Yang et al. 2016; Ahmed and Bagherzadeh 2018; Bagherzadeh and Khatchadourian 2019) due to its simplicity and precision, but it still requires the manual identification of appropriate tags. However, we cannot

Fig. 4 A Stack Overflow post (ID: 3226374) with incomplete tags

rely upon this approach alone as a reliable tagging system does not exist for GitHub issues. Furthermore, tagging itself is unreliable as it is performed by the user, who can submit incorrect or insufficient labels. For example, Fig. 4 depicts a Stack Overflow post which is not tagged with any security tag despite stimulating discussion about cross-site scripting and phishing.

Content-based filtering examines the actual content of a post to determine its relevance to a particular category through a set of keywords. If a post contains a lot of keywords which are representative of a particular topic (i.e., security), then the entire post can be considered as related to this topic. Content-based filtering has also been used in previous works when tag-based filtering could not be applied (Pletea et al. 2014; Zahedi et al. 2018) or to account for the shortcomings of tag-based filtering (Le et al. 2020). However, as keyword matching can produce false positives, content-based filtering requires threshold values to determine a post as relevant or not. For example, we cannot consider any post containing the word *cookie* as relating to web development, as this word may instead be referring to the tasty baked treat. Thus, we use unique count (the number of unique keywords in a post) and ratio (the ratio of keywords to the total number of words in a post) thresholds to ensure discussion relevance.

We use both the tag-list and keyword-list proposed by Le et al. (2020), as they are some of the most extensive security keyword lists from the previous literature. Our tag list consists of the tags: *security, cryptography, csrf, passwords, sql-injection* and *xss*, with the *security* tag being sub-string matched and all others exact matched. Our keyword list originally contained 234 security keywords, which we updated with manually identified missing variations to a total of 288 keywords.[9]

For content-based filtering all keywords and text are stemmed before keyword matching is applied. We consider different matching forms (i.e., American/British and with/without hyphen/space/stemming) to cope with various spellings. To reduce false positives we performed exact matching for short (three-character) keywords (e.g., md5) and sub-word matching otherwise (Le et al. 2020).

We empirically identified the suitable content thresholds for content-based filtering. For Stack Overflow, we obtained the content thresholds from the median values of the security tag filtered dataset; unique count = 4, ratio = 0.115. For GitHub, we first obtained all posts that contain one or more of the security keywords in the title, similar to the method used by Zahedi et al. (2018) and Pletea et al. (2014), and then manually examine a sample of 385 of these issues. Of this sample, only 70% of the posts were able to be confidently labelled as security-related by the authors, highlighting the need for stricter content thresholds. A range of threshold values were tested until a sample was deemed as sufficiently security related (>95% security-related); which uses a threshold of unique count = 2, ratio = 0.061.

Finally, we form our dataset of security related discussions for each language from each data source. For Stack Overflow, we used the union (non-overlapping) of the tag-based filtering set and content-based filtering set. For GitHub, we only used the content-based filtering set as previously stated.

We manually examined a sample of documents to confirm the content and security relation of the datasets.[10] We selected 385 documents from each source, for a confidence level of 95% and a confidence interval of 5% (Cochran 2007). For Stack Overflow 100% of the examined posts have a relation to security. For GitHub, 95% of the issues are considered as security relevant, while the other 5% contain unclear content or no reference to security.

---

[9]https://github.com/RolandCroft/Language-Security-Challenges/tree/master/Appendix/security_keywords.txt

[10]https://github.com/RolandCroft/Language-Security-Challenges/tree/master/Appendix/dataset_samples

For RQ1 we examine the collected data for all 50 languages. However, for subsequent research questions we limit our study to the most popular programming languages which have a substantial security consideration, to ensure that there is a sufficient amount of data to analyse and provide a better comparison. We select these languages as any language with a statistically significant number (>384) (Cochran 2007) of security related discussion posts on both Stack Overflow and GitHub. Hence, we only perform our in-depth analysis of the security challenges on a refined subset of 15 of the most dominant programming languages, shown in Table 4.

3.3 Manual Discussion Analysis

To address RQ2, we perform manual analysis of our data to determine the nature and purpose of the discussions. As the entirety of our dataset is naturally too large to manually examine, we take a statistically significant sample (confidence level 90% +/-10%) (Cochran 2007) of 68 posts per language for each data source (n=1904). We then manually categorize the purpose of each discussion based on the content of the question or issue. To define and categorize discussion intent, we adopt the curated taxonomy of question categories curated by Beyer et al. (2020). This taxonomy consists of seven question categories, as defined by Beyer et al. (2020) in Table 3.

As GitHub issues discussions do not strictly follow the question/answer format of Stack Overflow, we extend the taxonomy slightly to additionally account for the intent of GitHub user discussions. Firstly, GitHub issues related to Bug Fixes or inclusion of missing critical functionality are classified with the *Errors* category, as these issues are discussing fixes for software errors. Secondly, GitHub issues which introduce new or additional functionality (i.e., updates), are classified as *Review* as these issues are requesting for review, feedback or approval of code. Otherwise, we follow the same manual classification technique as described by Beyer et al. (2020). A post may relate to more than one category depending on its content.

Table 3 The seven question categories as defined by Beyer et al. (2020)

Category	Definition
API Usage	"The posts falling into this category contain questions asking for suggestions on how to implement some functionality or how to use an API. The questioner is asking for concrete instructions."
Discrepancy	"The posts of this category contain questions about problems and unexpected behavior of code snippets whereas the questioner has no clue how to solve it."
Errors	"Similar to the previous category, posts of this category deal with problems of exceptions and errors. Often, the questioner posts an exception and the stack trace and asks for help in fixing an error or understanding what the exception means."
Review	"Questioners of these posts ask for better solutions or reviewing of their code snippets. Often, they also ask for best practice approaches or ask for help to make decisions, for instance, which API to select."
Conceptual	"The posts of this category consist of questions about the limitations of an API and API behavior, as well as about understanding concepts, such as design patterns or architectural styles, and background information about some API functionality."
API Change	"These posts contain questions that arise due to the changes in an API or due to compatibility issues between different versions of an API."
Learning	"In these posts, the questioners ask for documentation or tutorials to learn a tool or language. In contrast to the first category, they do not aim at asking for a solution or instructions on how to do something. Instead, they aim at asking for support to learn on their own."

Two of the authors manually categorized each post to the above question categories. The authors first independently labelled 100 posts (50 from each source) to ensure consistent labelling. We calculated the Cohen's kappa coefficient (Cohen 1960) to determine the inter-rater agreement of the two authors, by calculating the kappa score for each label separately and then averaging scores. An overall kappa score of 0.65 was achieved on the initial set, implying substantial inter-rater agreement. Any initial disagreements in labelling were resolved through discussion. The authors then individually labelled the remaining posts.

3.4 Topic Modelling

We used topic modelling to identify the security discussion topics in our data sources due to its wide use on similar datasets and in related studies for both Stack Overflow (Yang et al. 2016; Barua et al. 2014; Zou et al. 2015; Allamanis and Sutton 2013) and GitHub (Zahedi et al. 2018; Rahman and Roy 2014). LDA exhibits the ability to form models without the need for labelled training data or pre-defined taxonomies. This suits our needs as Stack Overflow and GitHub discussions are largely unlabelled, and these informal discussions do not align with formal security taxonomies.

We used the Python *Gensim* library for the implementation of the LDA algorithm.[11] As Stack Overflow and GitHub discussions are naturally noisy (Barua et al. 2014; Zahedi et al. 2018), the pre-processing steps in Fig. 5 are applied. We use the union of the *Sklearn* and *NLTK* stopword lists (Nothman et al. 2018), and the *NLTK* Porter-Stemmer Algorithm.

Treude and Wagner (2019) show that corpora sampled from different languages of GitHub and Stack Overflow have varying characteristics and require different parameter configurations. Hence, we consider each language-specific security related dataset from each data source as a separate corpora, and trained the LDA models individually. In a related study, Han et al. (2020) also adopt a similar approach that they title as Balanced LDA. This method also removes any potential influence from the varying number of documents in each corpora.

Topic model quality heavily relies on appropriate hyper-parameter tuning (Agrawal et al. 2018). Hence, we optimized the following three LDA hyper-parameters via grid search: the number of topics k, the a-priori belief of each topic probability α, and the a-priori belief of each word probability β. For k, all values in the range 1 to 30 are tested. For α and β, we tested the default configuration ($\alpha = \beta = 1/k$), the automatically optimized configuration by *Gensim*, and the standard values from previous literature of $\alpha = 50/k$ and $\beta = 0.01$ (Ahmed and Bagherzadeh 2018; Bagherzadeh and Khatchadourian 2019; Yang et al. 2016). A model is trained with every combination of settings for each corpora.

Topic coherence measures the semantic similarity of high probability words in a topic (Mimno et al. 2011). This metric is selected for model evaluation, due to its ability to predict topic interpretability (Mimno et al. 2011) and its prevalence in related studies (Ahmed and Bagherzadeh 2018; Wan et al. 2019). The optimal model is chosen as the model with the highest coherence value. In the event of similar coherence values, the lower number of topics is chosen as it was empirically found that higher topic numbers produce more fine-grained and less comparable topics. The optimal number of topics for each corpora is shown in Table 4.

From the output of LDA, we obtain a set of K topics ($z_1, ..., z_k$) from each of our N corpora ($c_1, ..., c_n$). The symbols used to represent the topic attributes are summarized in

---
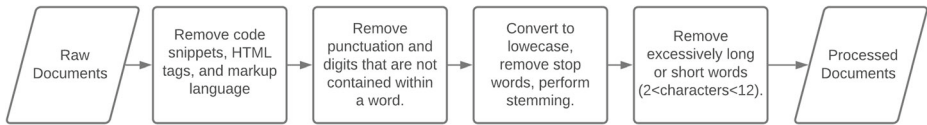
[11]https://radimrehurek.com/gensim/

Fig. 5 Pre-processing steps of Stack Overflow and GitHub documents

Table 5. Each topic is a distribution over the unique keywords of each corpora. For each post d in the corpora we also obtain a topic probability vector which indicates the proportion of words that come from each of the K topics (Blei et al. 2003). We denote the probability of a particular topic z_k for a particular post d_i as $\theta(d_i, z_k)$. Note that $\forall i, k : 0 \leq \theta(d_i, z_k) \leq 1$ and $\forall i : \sum_1^k \theta(d_i, z_k) = 1$.

Using the topic word distribution and topic probability vectors, topic labels are then assigned through collaboration of two of the authors. The authors assign topic categories using an open card sorting method (Fincher and Tenenberg 2005), in which the authors generate a dendogram of the items using self-generated labels. The authors first use the topic words to infer the label based on their own security expertise and judgement. The authors then examine documents in order of $\theta(d_i, z_k)$ (document-topic probability) to validate the inferred label. The authors examine documents until they are confident with the label; at least 10 documents were examined for each topic (provided the topic has more than 10 associated documents). If the authors are unconfident with the label after an examined portion of posts, a larger portion is examined to help generate a new label. Topics that do not implicitly relate to security or are too ambiguous to label are labelled as *Other*.

The first two authors first independently examined all topics to familiarize themselves with the data and gain an understanding of the potential topics and themes. During this process, each author tentatively assigned labels to each topic. The first two authors then

Table 4 The number of security-related Stack Overflow posts or GitHub issues and the optimal number of topics for each corpora

Programming Language	# Stack Over-flow Posts	# Stack Over-flow Topics	Stack Over-flow Model Coherence	# GitHub Issues	# GitHub Topics	GitHub Model Coherence
Java	35576	17	0.516	13647	14	0.538
PHP	32191	17	0.498	20398	19	0.502
JavaScript	22624	22	0.461	16246	17	0.489
C#	22151	23	0.485	11892	26	0.495
Python	11353	18	0.515	15368	20	0.566
Shell	5807	23	0.443	6395	11	0.567
C/C++	2641	13	0.472	16794	19	0.545
Ruby	2514	9	0.450	14537	11	0.535
Objective-C	2093	3	0.439	2472	14	0.446
Swift	1859	21	0.428	2124	7	0.522
PowerShell	1782	7	0.384	4112	3	0.584
Perl	942	11	0.442	2188	2	0.582
TypeScript	856	8	0.437	10227	24	0.526
Go	809	3	0.402	24288	13	0.507

Table 5 Topic attribute symbols

Symbol	Meaning	Specifics
(z_k)	Topic k	A security topic.
(c_j)	Corpora j	A security related programming language dataset from Stack Overflow or GitHub.
(d_i)	Document i	A Stack Overflow post or GitHub issue.
$\theta(d_i, z_k)$	Document-Topic Probability	The probability of a document belonging to a topic.

collaboratively discussed and refined the topic categories. Finally, to reduce potential individual researcher bias during this manual process, the final topic labels are assigned collaboratively by the first two authors in unison. The first author has over three years of experience doing security related research in academia, whereas the second author has over 13 years of security experience working in industry. Disagreements are resolved through discussion.

To assess the potential subjectivity of the topic labelling process, we used an approach similarly performed in other works (Hata et al. 2019). Once, the final topic categories had been defined, the first two authors independently annotated a random sample of 30 topics. We then calculated the Cohen's kappa coefficient (Cohen 1960) to determine the inter-rater agreement of the two authors, by calculating the kappa score for each topic category separately and then averaging scores. An overall kappa score of 0.79 was achieved on the initial set, implying substantial inter-rater agreement. Based on this encouraging result, we considered the topic labelling process to be sufficiently robust to potential researcher bias.

After all labels have been generated, topic grouping and categorisation is then conducted using a bottom-up categorisation approach: beginning with the initially assigned specific topic labels, sub-categories of topics are then identified, which are then further grouped into final high level categories. Whilst individual topics and sub-categories can potentially relate to numerous groupings, we assign them to the most relevant category. To reduce the dimension of the data, we only consider (sub-)categories for analysis, and group individual topics based on this.

We provide a full example of the labelling process here. One of the produced topics for Java contains the keywords *user, authentication, application, ldap, credentials, access, login*. Based on the context of these keywords we interpret the topic as being related to user login and authentication. We confirm this interpretation by then examining associated posts for this topic to see if they fit the context, e.g., "*How can I allow my users to login with Windows credentials?*" *(SO, 13927479)*. Once we have completed this process, we then consider all topics, regardless of source and programming language, to create our topic taxonomy. For each topic, we first assign a category based on other similar topics that have manifested. Topics that have many other semantically similar topics become a category, and are assigned a label that broadly captures the definition of all the topics. This example topic we assign the Authentication category. Then once again after we have assigned all topics to a category, we finally group the categories into higher-level definitions; in this case, Access Control.

The security taxonomy that emerged from our data is presented in Table 6. We also show the number of individual topics that was mapped to each category.

We define the dominance of a topic in a particular corpus via the *Topic Share* metric (Barua et al. 2014). The topic share value measures the proportion of posts in a corpus (c_j)

Table 6 Our identified security taxonomy and topic mapping

Category	Sub-Category	# Topics
Cryptography	Encryption	42
	Encoding	12
Access Control	Authentication	40
	Tokens	25
	Authorization	10
Network Security	Digital Certificates	26
	Client/Server	23
	File Transfer	7
Data Security	Password/Data Storage	23
	Digital Signature	7
Software Vulnerabilities	Network Exploits	18
	Mitigation	5
	Memory Management	6
	Race Condition	4
Secure Development	Implementation	57
	Testing	25
	Libraries/Configuration	39
	GitHub/Reports	15
Other		11
Total		395

which contain a specific topic (z_k):

$$share(z_k, c_j) = 100 \times \frac{1}{|c_j|} \sum_{d_i \in c_j} \theta(d_i, z_k) \qquad (1)$$

To evaluate the temporal trends of particular topics within the corpora for RQ4, we utilize the topic impact metric, as used in similar works (Panichella et al. 2013; Rosen and Shihab 2016; Wan et al. 2019). The impact of a topic z_k in month m is defined as:

$$impact(z_k, m) = \frac{1}{|D(m)|} \sum_{d_i \in D(m)} \theta(d_i, z_k) \qquad (2)$$

where $D(m)$ is the set of documents within the corpus that were posted in month m. For example, for a month in which 3 issues were posted which each relate to the same topic with probabilities: 0.3, 0.4, 0.8, then the topic impact value of that topic for that month would be: $(0.3 + 0.4 + 0.8)/3 = 0.5$. This gives us an indication of the each topic's significance per month, for the time range of January 2009 to May 2020. The topic impact score can be considered as the topic share value over time.

3.5 Topic and Language Characteristics

We obtain various metrics of the posts and users for each language and its topic categories to reveal aspects of their characteristics. These metrics inform us of the relative significance

of the challenges when they occur, and the nature in which they are resolved by online communities.

For RQ5, we focus on the attributes of the documents that make up the various topics and languages. As it is difficult to determine the document-centric metrics for partial topics via the topic share value, we cluster posts based on their topic probability vectors. We define the dominant topic for each post to be the topic with the highest probability for that document (Panichella et al. 2013; Rosen and Shihab 2016; Wan et al. 2019). This is formally defined as:

$$dominant(d_i) = z_k : \theta(d_i, z_k) = max(\theta(d_i, z_j)), 1 \leq j \leq K \quad (3)$$

The size of a topic ($|z_k|$) is the number of posts for which it is the dominant topic, i.e., $\sum max(\theta(d_i, z_k))$. To obtain the set of dominant topics for a category, we take the union of document sets for each topic which forms that category. Similarly, we can calculate the overall popularity or difficulty of a language through the set of all documents pertaining to that language.

For Stack Overflow, we use document-centric metrics inspired by previous works (Yang et al. 2016; Bagherzadeh and Khatchadourian 2019; Ahmed and Bagherzadeh 2018) to measure the popularity and difficulty.

The popularity of a given Stack Overflow topic category is calculated as follows:

$$Popularity_{SO} = \frac{1}{|T|} \sqrt[4]{\sum P1 \times \sum P2 \times \sum P3 \times \sum P4} \quad (4)$$

where for each document in that topic, P1 is the number of views, P2 is the score, P3 is the favorites, P4 is the number of comments, and $|T|$ is the number of documents in that topic category.

The difficulty of a given Stack Overflow topic category is calculated as follows:

$$Difficulty_{SO} = \sqrt[3]{\frac{|T|}{\sum D1} \times \frac{\sum D2}{|T|_{accepted}} \times \frac{|T|_{accepted}}{\sum D3}} \quad (5)$$

where for each document in that topic, D1 is whether the question has an accepted answer, D2 is the time in hours to receive an accepted answer and D3 is the number of answers per the number of views. As not all questions have answers or accepted answers, D2 and D3 are determined in relation only to posts that have an accepted answer. D1 and D3 have an inverse relationship to difficulty; a higher value indicates an easier topic.

To the best of our knowledge, ours is the first work to attempt to formally quantify the popularity or difficulty of GitHub Issues via their discussions. We adapt the metrics used for Stack Overflow to also be suitable for GitHub documents as well, but due to the differences in the sites metadata and activities we note that the conversion is not directly equivalent.

We define the popularity of a given GitHub topic category as follows:

$$Popularity_{GH} = \frac{1}{|T|} \sqrt[3]{\sum P1 \times \sum P2 \times \sum P3} \quad (6)$$

where for each document in that topic, P1 is the number of comments, P2 is the number of unique users who contribute to the issue and P3 is the number of 'up' reactions.[12] We use GitHub reactions as a metric due to their possible resemblance to the score of a Stack Overflow posts, and reactions are becoming increasingly commonplace in GitHub issue discussions (Borges et al. 2019).

---

[12]https://developer.github.com/v3/reactions/

The difficulty of a GitHub topic category is defined as:

$$Difficulty_{GH} = \frac{1}{|T|} \sqrt[2]{\sum D1 \times \sum D2} \tag{7}$$

where for each document in that topic, D1 is the time in hours to close that issue and D2 is the number of commits assigned to that issue. We find that there are a significant number of outliers for issue close time due to stale issues (Wessel et al. 2019), so for D1 we remove outliers using an outlier boundary of $1.5 * Interquartile Range$.

It is important to note that these metrics are different to the topic share metric measured in RQ3 and obtain a different ranking, which we confirm using the Kendall rank correlation coefficient (Knight 1966) at the 95% confidence level. Whilst the topic share indicates the overall prevalence of a topic, the popularity measures the interest and engagement of a post for that topic when it occurs.

For expertise, we adopt the metrics proposed by Tian et al. (2019) which identify the relative expertise of a user on either Stack Overflow or GitHub for a particular topic. The full equations for the expertise of a Stack Overflow or GitHub user can be seen in their paper (Tian et al. 2019). In our case the topic of expertise is *security*. We adopt this particular approach as it considers various important user attributes and platform specific information.

For Stack Overflow, the expertise of a given user is calculated through a combination of their profile performance (reputation and profile views) and their answer performance (answer scores and score/favorites/views of questions they are answering). To relate the expertise to a specific topic, only the answer performance for questions of that topic are considered. Hence, we obtain the user answer performance for the security related posts. To extend the generalizability of security knowledge, we also consider general security related posts which are collected using the same heuristics as the dataset created by Yang et al. (2016).

For GitHub, the expertise of a user is determined through their contribution to significant repositories. This is calculated by the frequency of a user's commits, and the total commits and watchers of a repository. To make the expertise value topic specific, the topic weight of the repository is also determined. We altered the weight formula proposed by Tian et al. (2019) slightly, to better represent a user's security contribution, and calculate it as:

$$Weight = \frac{numUserSecurityComments}{numSecurityComments} \tag{8}$$

As we intend to investigate how challenges are resolved, only the expertise of users who provide answers to security related discussions are considered, either through the accepted answer on Stack Overflow, or a comment on a GitHub issue.

The individual values of these metrics (popularity, difficulty and expertise) are not indicative on their own, so we only consider the normalized values for the purposes of comparison in RQ5 and RQ6.

4 Results and Analysis

4.1 RQ1: What is the Rate of Security Discussion Amongst Programming Languages on Stack Overflow and GitHub?

The selected languages and their rate of security discussion is shown in Table 7. Related literature has reported the frequency of security discussion on Stack Overflow to be between

Table 7 The rate of security related discussion (%) for each programming language for each source

Programming Language	Stack Overflow			GitHub			Difference*	Average
	Security Posts	Total Posts	Security Rate	Security Issues	Total Issues	Security Rate		
Shell	5807	152203	**3.82**	6395	458529	**1.39**	+2.42	**2.60**
PHP	32191	1361990	**2.36**	20398	1157572	**1.76**	+0.60	**2.06**
PowerShell	1782	82460	**2.16**	4112	266222	**1.54**	+0.62	**1.85**
Go	809	46051	**1.76**	24288	1448882	**1.68**	+0.08	**1.72**
Erlang	126	8816	**1.43**	1495	78827	**1.90**	−0.47	**1.66**
Java	35576	1689665	**2.11**	13647	1234379	**1.11**	+1.00	**1.61**
Puppet	49	3762	**1.30**	198	11734	**1.69**	−0.38	**1.49**
Perl	942	65995	**1.43**	2188	145351	**1.51**	−0.08	**1.47**
C/C++	2641	330139	**0.80**	16794	871251	**1.93**	−1.13	**1.36**
Ruby	2514	214226	**1.17**	14537	987441	**1.47**	−0.30	**1.32**
C#	22151	1432299	**1.55**	11892	1118533	**1.06**	+0.48	**1.30**
Groovy	329	25317	**1.30**	1259	119471	**1.05**	+0.25	**1.18**
Python	11353	1536950	**0.74**	15368	963333	**1.60**	−0.86	**1.17**
VB.NET	1332	131530	**1.01**	55	4655	**1.18**	−0.17	**1.10**
Elixir	82	7737	**1.06**	1145	108948	**1.05**	+0.01	**1.06**
JavaScript	22624	2016932	**1.12**	16246	2080976	**0.78**	+0.34	**0.95**
Objective-C	2093	291030	**0.72**	2472	270107	**0.92**	−0.20	**0.82**
Kotlin	298	39886	**0.75**	2011	236977	**0.85**	−0.10	**0.80**
Lua	111	17017	**0.65**	1340	143203	**0.94**	−0.28	**0.79**
CoffeeScript	68	9767	**0.70**	933	105551	**0.88**	−0.19	**0.79**
Swift	1859	285658	**0.65**	2124	248266	**0.86**	−0.20	**0.75**
Clojure	82	16297	**0.50**	1236	123395	**1.00**	−0.50	**0.75**
Fstar	0	20	**0**	39	2642	**1.48**	−1.48	**0.74**
TypeScript	856	114778	**0.75**	10227	1455275	**0.70**	+0.04	**0.72**
Assembly	111	34129	**0.33**	252	27049	**0.93**	−0.61	**0.63**
QML	20	9094	**0.22**	85	8265	**1.03**	−0.81	**0.62**
Dart	194	29978	**0.65**	1253	212221	**0.59**	+0.06	**0.62**
Rust	44	16243	**0.27**	4568	498734	**0.92**	−0.65	**0.59**
Pascal	10	2156	**0.46**	187	25911	**0.72**	−0.26	**0.59**
Groff	0	56	**0**	77	6783	**1.14**	−1.14	**0.57**
Haskell	105	43795	**0.24**	1378	165280	**0.83**	−0.59	**0.54**
Vala	3	876	**0.34**	166	23085	**0.72**	−0.38	**0.53**
Scala	327	97971	**0.33**	3203	456422	**0.70**	−0.37	**0.52**
TSQL	221	62276	**0.35**	246	47713	**0.52**	−0.16	**0.44**
OCaml	5	6206	**0.08**	698	98536	**0.71**	−0.63	**0.39**
Smalltalk	4	1563	**0.26**	63	12573	**0.50**	−0.25	**0.38**
VimL	0	41	**0**	255	121604	**0.72**	−0.72	**0.36**
D	3	2540	**0.12**	259	47265	**0.55**	−0.43	**0.33**
DM	0	49	**0**	736	121218	**0.61**	−0.61	**0.30**
F#	17	14631	**0.12**	211	46838	**0.45**	−0.33	**0.28**

Table 7 (continued)

Programming	Stack Overflow			GitHub			Difference*	Average
Language	Security Posts	Total Posts	Security Rate	Security Issues	Total Issues	Security Rate		
Scheme	2	7196	**0.03**	57	10966	**0.52**	−0.49	**0.27**
Raku	0	395	**0**	41	9101	**0.45**	−0.45	**0.23**
Prolog	9	11246	**0.08**	3	883	**0.34**	−0.26	**0.21**
ELisp	3	3735	**0.08**	415	132570	**0.31**	−0.23	**0.20**
R	320	346073	**0.09**	487	166897	**0.29**	−0.20	**0.19**
Coq	2	2053	**0.10**	14	7658	**0.18**	−0.09	**0.14**
Julia	6	6757	**0.09**	255	149348	**0.17**	−0.08	**0.13**
Fortran	1	10903	**0.01**	51	34087	**0.15**	−0.14	**0.08**
MATLAB	67	88285	**0.08**	3	18230	**0.02**	+0.06	**0.05**
Total	150285	11358546	**1.32**	235141	20832355	**1.13**	0.19	**1.23**

*A + symbol indicates the increase in rate of discussion on Stack Overflow compared to GitHub, and a - symbol indicates a decrease in rate of discussion on Stack Overflow compared to GitHub.

1% (Yang et al. 2016) and 2% (Le et al. 2020), and the rate of security discussion on GitHub to be between 3% (Zahedi et al. 2018) and 10% (Pletea et al. 2014). For our dataset, we observe the average rate of security discussion on Stack Overflow to be 1.32% which is inline with previous work. However, for GitHub we observe a lower average rate of 1.13%. Using the same data filtering methods as Zahedi et al. (2018) on our dataset, we identify a rate of 3.38%, but we occasionally found these posts to lack relevance and focus. Hence, we used stricter keyword count thresholds for content-based filtering to ensure document quality, which results in the lower rate of discussion.

Shell-based languages (i.e., Shell and PowerShell) have some of the highest rates of security discussion (2.6% and 1.85%), which is likely due to their stronger coupling with the underlying operating system. Languages commonly used for web development (i.e., PHP, Ruby and JavaScript) also have slightly higher rates of security discussion (2.06%, 1.32% and 0.95%) due to the increased security needs of these highly connected software systems. Conversely, languages oriented towards numerical analysis (Scientific Programming Languages, i.e., R, Julia and MATLAB) have exceedingly low rates of security discussion with less than 0.2% each.

General purpose languages typically have a higher average rate of security discussion, and more obscure languages have a lower average rate of security discussion. Similarly, several of the languages with the highest rates of security discussion are widely used (i.e., Go, Java, C/C++, C# and Python). However, these factors may only be because of their naturally higher rate of overall discussion and use.

Interestingly, languages which require manual memory management (i.e., C/C++, Objective-C, Pascal, Assembly and Fortran) do not necessarily have a higher rate of security discussion compared to languages which are considered"memory-safe". Programs written in these languages are especially prone to extreme and unexpected input data via buffer overflow and other related vulnerabilities. However, C/C++ exhibits the highest frequency of security discussion for GitHub issues (1.93%).

Thirty nine out of the fifty examined languages exhibit a cumulative rate of discussion below the average ($<1.23\%$). This might be an indication of a general lack of knowledge, prioritisation, or consideration within these programming languages.

Using one way analysis of variance (ANOVA) (Howell 2012) we confirm that there is no significant correlation between the rate of security discussion and the type class, compilation class or memory class paradigms of a programming language. We also confirm that there is no significant correlation between the creation date of a programming language and its rate of security discussion, using the Kendall Tau Coefficient (Knight 1966).

> **RQ1 Summary:** The average frequency of security discussion is around 1% for both Stack Overflow and GitHub. Shell-based and Web-oriented languages exhibit the highest rates of discussion (2.6% and 2.1% for Shell and PHP respectively), whereas scientific programming-based languages exhibit the lowest (0.1% for R, Julia and MATLAB).

4.2 RQ2: What is the Intention Behind Security Discussions for Different Programming Languages?

Using a manually labelled sample, we examine the types of posts using the question taxonomy of Beyer et al. (2020), to determine the purpose and intent of the security discussions for each language. The manually labelled and annotated samples as well as the full category distributions can be viewed in our online appendix.[13]

For the seven proposed question categories, we find that the *Learning* category was extremely infrequent in our dataset. Only 3 out of 952 posts were assigned this label for Stack Overflow, and none for GitHub, so we exclude this category from our analysis. We found the nature of the discussions in our dataset to be more functionally orientated. Similarly, the *Conceptual* category was also under-represented. As most posts discussed implementation related tasks (especially for GitHub), we found the separation between this category and *Review* to be indistinct. We have hence merged the two categories to increase label consistency.

We present the overall trends for each category in Fig. 6. For Stack Overflow, *API Usage* and *Errors* are the most prominent reasons for discussion as users are commonly asking for assistance in solving a problem. For GitHub, there is a much higher proportion of *Review* questions, as users require approval for proposed changes. We next inspect the difference in distribution for individual programming languages.

We first consider whether there is a difference in the distribution of question categories for different programming languages. Using a Chi-Squared Test of Independence (Pearson 1900), we find that there is an association between the programming language and the category of a post, with $p <0.01$ for both Stack Overflow and GitHub. Using this knowledge, we then examine how the distribution of question/issue types varies to the standard distribution of each category. Figure 7 displays the standard score (difference in standard deviations to the mean) for the distribution of each question/issue category for each language. Languages with green bars consist of a higher than average proportion of posts with that type of question/issue, and a lower proportion for red bars.

---

[13]https://github.com/RolandCroft/Language-Security-Challenges/tree/master/Manual_Analysis
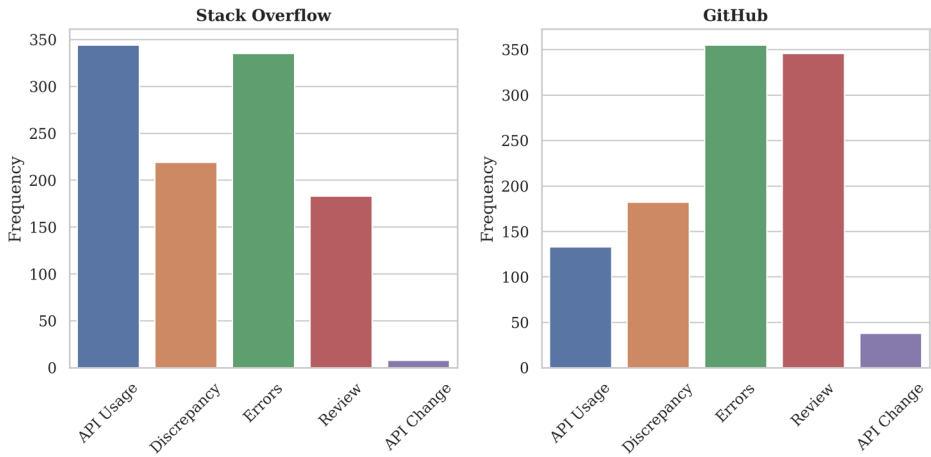
Fig. 6 The overall discussion categories for each source

　　Through these question/issue types, we can infer the motivations and purpose of the security discussions for each language, and consequently identify the potential deficiencies:

–　*API Usage:* This category relates to the posts discussing implementation or use of a specific API. We find that posts of this nature indicate that security is generally hard for a particular language. Implementation is intensive or lacking documentation; hence users need support through discussion. For instance, "*how to add interceptor in spring*" *(SO, 8033100)* or "*clarification of documentation for KeePassXC entry protection*" *(GH, asbru-cm/asbru-cm/issues/565).*

–　*Discrepancy:* This category involves the posts where users encounter unexpected behaviour. The user wants to know what or why something does not work. We observe that the posts of this nature imply a lack of familiarity with security flaws and issues. Perhaps due to a lack of experience, the users have little understanding or knowledge of the security concept or functionality, which may lead to confusion. For instance, "*PHP Login script only works with usernames known to mysql?*" *(SO, 56138399).*

–　*Errors:* These posts are similar to those of the *Discrepancy* category, but explicitly involve exceptions or errors. For GitHub, this category is extended to also consider issues describing how to fix these errors. This category implies that implementation of security is difficult or error-prone. Users make mistakes or miss critical functionality, which needs to be fixed. For example, " *'Unauthorized' error in .net webservices developed in C#*" *(SO, 1848508).*

–　*Review:* This category is merged with the *Conceptual* category. These posts attempt to review or understand concepts and functionality, for the purpose of improvement or decision making. For GitHub, this is extended to include code improvement and updates, as users are seeking review and validation. Questions of this nature may indicate that a user's current implementation and security expertise are not sufficient to meet security requirements; they are not confident enough in their secure coding skills to ensure security. For instance, "*Is this sufficient security for user input in PHP?*" *(SO, 11003851).*

–　*API Change:* Posts for this category discuss compatibility issues and discrepancies between API and software versions. This implies that security issues
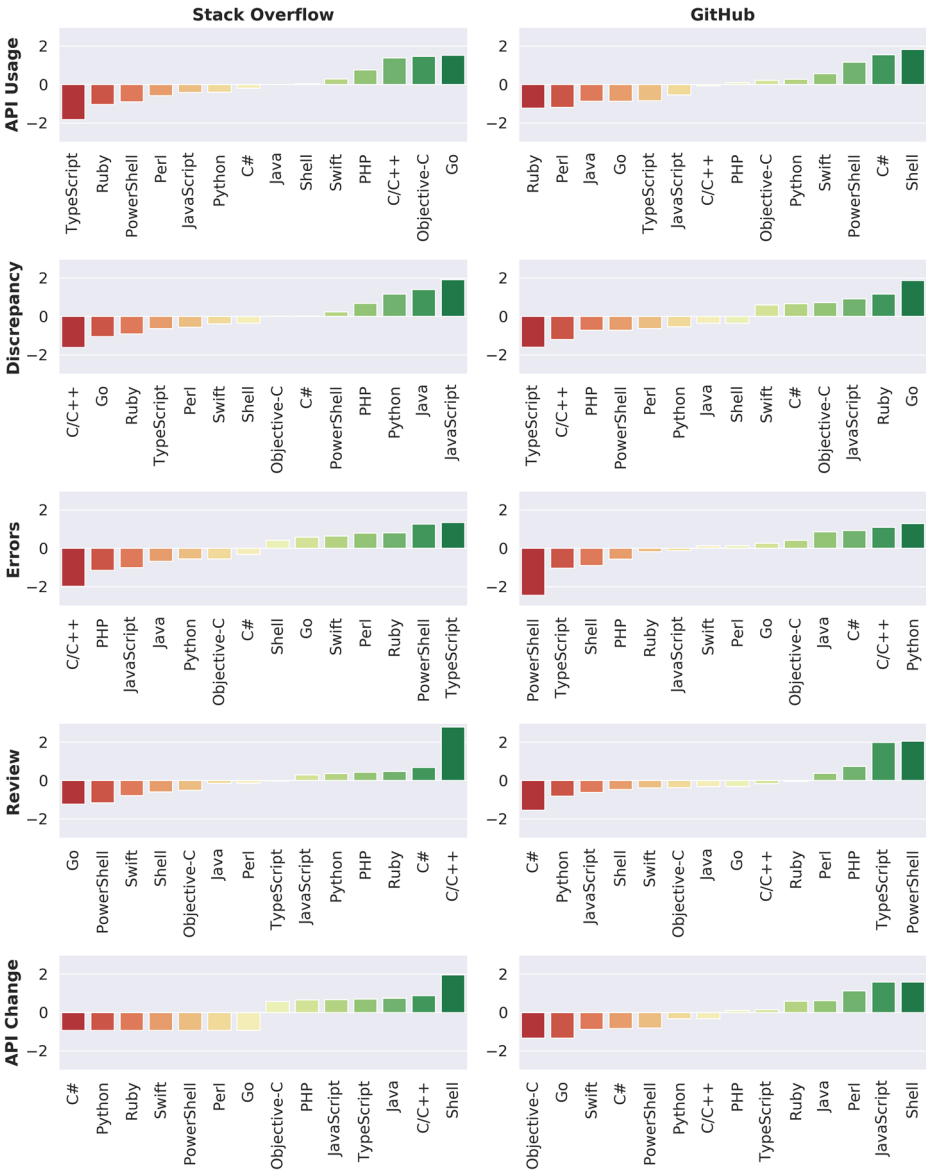
Fig. 7 The variance in standard deviation of the distribution of discussion categories for each of the selected programming languages

arise from external software, either from package vulnerabilities or deprecated security features. For example, "*Chrome drops support for added credentials*" *(GH, Codeception/Codeception/issues/4384)*, or "*DeprecationWarning: crypto.DEFAULT_ENCODING is deprecated*" *(GH, textlint/textlint/issues/620).*

Security discussions for C/C++ are more heavily oriented towards *Review* than other languages. We observe that developers are much more conscientious of security for these

languages. Questions often involve explanation of a vulnerability or attack (e.g., "*Why check if pointer is NULL after using it?*" *(SO, 26969565)*, "*How to use Format String Attack?*" *(SO, 27018864)*), improvement of code (e.g., "*In C, what is a safer function to use than strtrns?*" *(SO, 9264942)*), or security confirmation (e.g., "*Do the openssl X509_verify_cert() verifies the signature in the certificate?*" *(SO, 10495903)*). This implies that developers are aware of the secure coding requirements for C and C++ (Seacord 2005), but require knowledge support. To account for this variation, there is a lower than average distribution of questions for *Errors* and *Discrepancy*.

PowerShell issues are less oriented towards *Errors* than other languages. They are also more oriented towards *Review*, but neither of these trends are reflected in Stack Overflow questions. Upon inspection, we found that a large proportion of GitHub issues related to PowerShell involved updates or feedback on documentation. This is due to the inclusion of Microsoft Documentation repositories,[14] which are written in PowerShell, but primarily oriented to documentation.

TypeScript also has a high distribution of issues about *Review*, as discussions were most commonly about security code/feature updates and improvement, e.g., "*Improve the settings sync flows with authentication*" *(GH, microsoft/vscode/issues/94766)*. This indicates that security in TypeScript is intensive and requires a lot of implementation.

Shell discussions are more oriented towards *API Change* on both Stack Overflow and GitHub. We find that developers typically use Shell scripts to install, call or update security related packages and components. Hence, these posts typically discuss version issues (e.g., "*Issue with Debian 10 <tls-crypt >empty*" *(GH, angristan/openvpn-install/issues/523)*) or updates (e.g., "*update oraclelinux for openssl*" *(GH, docker-library/official-images/issues/4957)*, "*Update the OpenVPN mirror GPG key and fingerprint*" *(GH, StreisandEffect/streisand/issues/754)*).

> **RQ2 Summary:** There is a correlation between programming language and the purpose of the security discussions. C/C++ developers are much more conscientious of security but require knowledge support. TypeScript issues are often about security updates and improvements. Security challenges for Shell are motivated through use of external components.

4.3 RQ3: What are the Major Developer Security Discussion Topics?

To address this research question, we first create a taxonomy of the security discussion topics, as identified via the topic models. The taxonomy is displayed in Fig. 8. This taxonomy is based on the dominant and emergent discussion topics in our dataset. It should be noted that this taxonomy is not exhaustive, nor does this encompass all elements of cybersecurity. The taxonomy is focused around the content of our dataset, security related development issues, rather than concepts and knowledge. Furthermore, if a category is absent for a particular language, it does not necessarily mean that the discussion is non-existent; it may just be overshadowed by other topics.

Labels are not assigned to topics which do not implicitly relate to security or are too ambiguous to label. Eleven out of the 395 total individual topics of all languages and sources
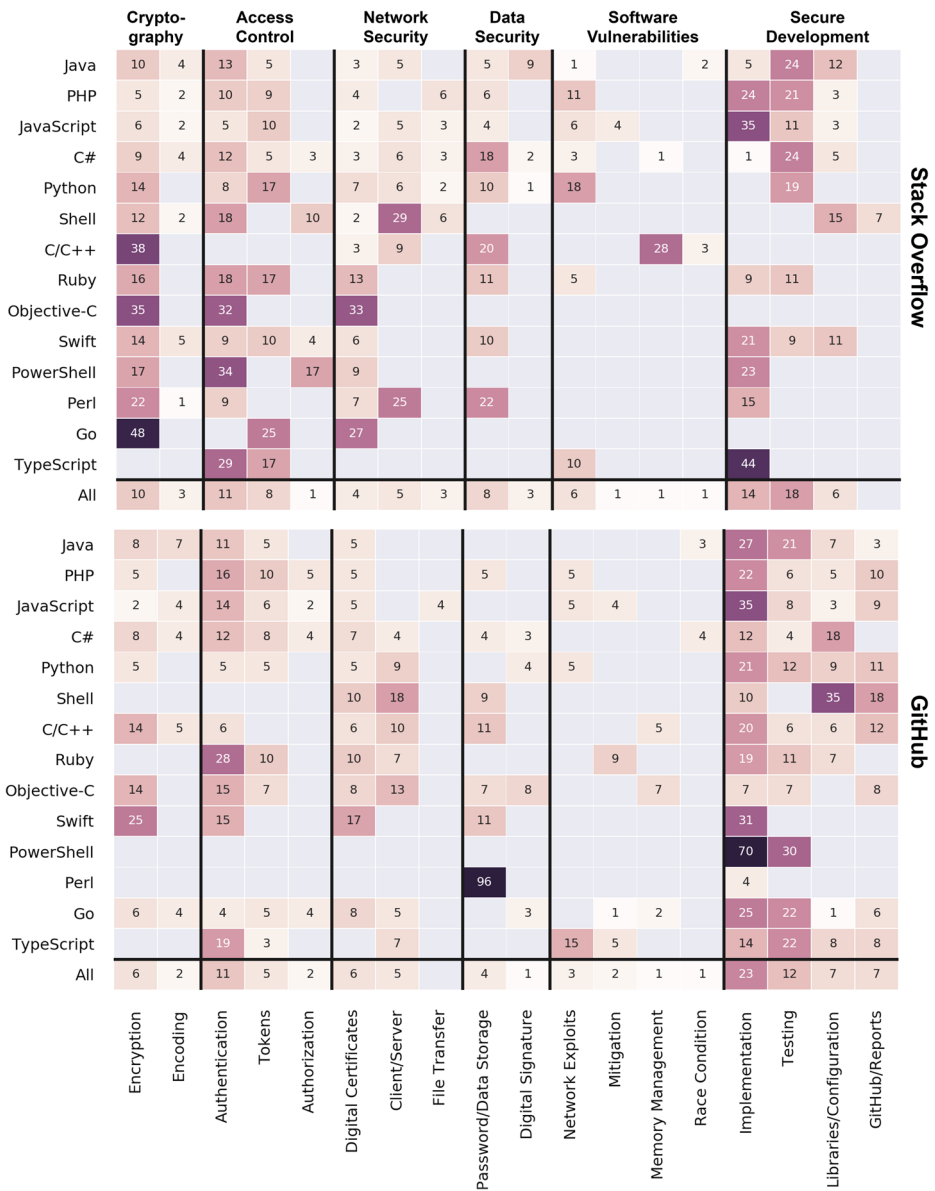
---

[14] https://github.com/MicrosoftDocs

Stack Overflow

	Cryptography		Access Control			Network Security			Data Security		Software Vulnerabilities				Secure Development			
	Encryption	Encoding	Authentication	Tokens	Authorization	Digital Certificates	Client/Server	File Transfer	Password/Data Storage	Digital Signature	Network Exploits	Mitigation	Memory Management	Race Condition	Implementation	Testing	Libraries/Configuration	GitHub/Reports
Java	10	4	13	5		3	5		5	9	1			2	5	24	12	
PHP	5	2	10	9		4		6	6		11				24	21	3	
JavaScript	6	2	5	10		2	5	3	4		6	4			35	11	3	
C#	9	4	12	5	3	3	6	3	18	2	3		1		1	24	5	
Python	14		8	17		7	6	2	10	1	18					19		
Shell	12	2	18		10	2	29	6								15	7	
C/C++	38					3	9		20				28	3				
Ruby	16		18	17		13			11		5				9	11		
Objective-C	35		32			33												
Swift	14	5	9	10	4	6			10						21	9	11	
PowerShell	17		34		17	9									23			
Perl	22	1	9			7	25		22						15			
Go	48			25		27												
TypeScript			29	17							10				44			
All	10	3	11	8	1	4	5	3	8	3	6	1	1	1	14	18	6	

GitHub

	Cryptography		Access Control			Network Security			Data Security		Software Vulnerabilities				Secure Development			
	Encryption	Encoding	Authentication	Tokens	Authorization	Digital Certificates	Client/Server	File Transfer	Password/Data Storage	Digital Signature	Network Exploits	Mitigation	Memory Management	Race Condition	Implementation	Testing	Libraries/Configuration	GitHub/Reports
Java	8	7	11	5		5								3	27	21	7	3
PHP	5		16	10	5	5			5		5				22	6	5	10
JavaScript	2	4	14	6	2	5		4			5	4			35	8	3	9
C#	8	4	12	8	4	7	4		4	3				4	12	4	18	
Python	5		5	5		5	9			4	5				21	12	9	11
Shell						10	18		9						10		35	18
C/C++	14	5	6			6	10		11				5		20	6	6	12
Ruby			28	10		10	7					9			19	11	7	
Objective-C	14		15	7		8	13		7	8			7		7	7		8
Swift	25		15			17			11						31			
PowerShell															70	30		
Perl									96						4			
Go	6	4	4	5	4	8	5			3		1	2		25	22	1	6
TypeScript			19	3			7				15	5			14	22	8	8
All	6	2	11	5	2	6	5		4	1	3	2	1	1	23	12	7	7

Fig. 8 The heatmap of the topic share (%) for the identified topic sub-categories for the 15 selected programming languages for Stack Overflow and GitHub

were not assigned a label, and thus excluded from this study. The impact of this exclusion is minimal as the 11 topics each have a topic share value of less than 1.

The heatmap of the topic distribution for each programming language of each source is shown in Fig. 8, where each value is the percentage topic share. Cells are rounded to nearest integer, where cells with no number have a topic share value rounded to 0. As such, rows

do not necessarily total 100. It should be noted that the distribution of each programming language is limited to topics which actually appeared in the topic models. Thus, languages with less diversity tend to have higher topic share values. The overall topic share value for each data source (considering all documents of a data source as a single corpus, regardless of language) is given in the *All* row to give an indication of the overall topic distribution. An explanation and analysis of the taxonomy categories is below.

Category - Cryptography Cryptography relates to the techniques and practices for secure communication and information security. Cryptographic systems are a security primitive, essential for ensuring data confidentiality, data integrity, authentication and non-repudiation (Menezes et al. 2018). As such, this category exhibits some overlap with other categories, but the distinction lies in the focus of the techniques and methods, rather than the application and uses. Noticeably, TypeScript exhibits no Cryptography related discussions or challenges on both sources.

– **Encryption:** All discussion relating to encryption/decryption and relevant cryptographic algorithms, without any explicit application. For instance, "*How do you encrypt large files / byte streams in Go?*" *(SO, 49546567)* or "*Encrypted String is wrong in swift*" *(GH, krzyzanowskim/CryptoSwift/issues/774)*. This is one of the more prominent sub-categories for both Stack Overflow and GitHub, due to its diverse applications. Across the two sources, discussion is most dominant for C/C++, Objective-C, Swift and Go, with each of these languages exhibiting a quarter of their overall discussion for this topic. Topics for C/C++, Shell, JavaScript and Java have a bigger focus on the cryptographic algorithms rather than encryption itself (e.g., "*Deriving ECDSA Public Key from Private Key*" *(SO, 49204787)*). This is most noticable for C/C++ due to the large number of discussions for *OpenSSL*,[15] a cryptography toolkit. This indicates that use of cryptography is very strong for this language.
– **Encoding:** The encoding or decoding of data. Whilst encoding is not explicitly related to cryptography, it was found that the majority of discussions of this topic fell under the cryptography category. For instance, the post "*Could not parse base64 DER-encoded ASN.1 public key from iOS in Golang*" *(SO, 48761010)* discusses encoding a public key in Go.

Category - Access Control Access Control involves the methods, practices and techniques of restriction and permissive access to resources, and other access management processes. This is the second most prominent category, behind Secure Development. Challenges occur for all languages across either source.

– **Authentication:** Discussion relating to user validation, typically through login, accounts or credentials. We found the majority of topics and posts for this sub-category to relate to implementing a login system, e.g., "*Ruby on Rails: Devise - password change on first login*" *(SO, 13121356)*. As such, challenges are particularly prominent in languages oriented for app development (i.e., Web or Mobile development), like PHP, C#, Ruby, Objective-C, Swift, PowerShell, and TypeScript. The high overall topic share value for these discussions suggest that it is one of the most common security-related tasks that developers face.

---

[15]https://www.openssl.org/

– **Tokens:** Token-based authentication and other similar mechanisms, such as API tokens, cookies, and general session management. This sub-category is more frequent on Stack Overflow, for languages such as Python, Ruby, Go and TypeScript. API tokens such as OAuth token authentication are a prominent discussion point, e.g., *"Google oauth how Exchange code for access token and ID token" (SO, 22796143)*.

– **Authorization:** The process of assigning and checking user permissions to access a resource. This category is noticeably significant on Stack Overflow for Shell (10%) and PowerShell (17%) due to the use of methods such as Secure Shell (SSH) for connecting to remote servers (i.e., *"Validate SSH connectivity using SSH-keys" (SO, 38006170)*). For all languages, discussion of Authorization is less than that of other Access Control sub-categories.

Category - Network Security Network Security involves the practices required to enable secure communication and data transfer over a computer network.

– **Digital Certificate:** Secure communication through the use of digital certificates. One of the more prominent topics for Network Security, with the highest topic share values for Objective-C (33%) and Swift (17%) on Stack Overflow and GitHub respectively. These languages are both commonly used for iOS development, e.g., *"How do I accept a self-signed SSL certificate using iOS 7's NSURLSession" (SO, 30739149)*.

– **Client/Server:** Secure communication over a network to a server and/or client. This sub-category is most prominent for Shell (29% and 10% Topic Share for Stack Overflow and GitHub respectively), again due to the use of Secure Shell (SSH). For example, *"How do I access a WebService through an SSH tunnel?" (SO, 2289708)*. Besides Shell, this topic is also noticeably evident for Perl and C/C++. These languages are commonly utilized for socket programming. For instance, this issue *"net.socket:connect() - leaks memory on DNS lookup fail" (GH, nodemcu/nodemcu-firmware/issues/234)* is about a security problem for a socket connection in C.

– **File Transfer:** Secure transfer and access of files over a network, e.g., *"Security threats with uploads" (SO, 11061355)*. This topic is most prominent for PHP and JavaScript on Stack Overflow and GitHub respectively, but it is overall relatively infrequent.

Category - Data Security Data Security involves protecting digital data from attacks or breaches.

– **Password/Data Storage:** Secure storage of passwords or data, usually through cryptographic means of encryption or hashing. Discussions for this topic often relate to implementation of secure password stores, e.g., *"Hash and salt passwords in C#" (SO, 2138429)*. As such, this topic is noticeably evident in languages commonly used for app development. Interestingly, almost all of the security-related documents on GitHub for Perl (96%) are for this topic, as well as a large portion from Stack Overflow (22%). Aside from the higher proportion, we did not observe any distinct characteristics of the Perl posts in comparison to other languages.

– **Digital Signature:** Verifying the authenticity of data or documents through the use of digital signatures, e.g. *"Add script to verify signature" (GH, sparkle-project/Sparkle/issues/896)*. Due to the much higher topic share value for Password/Data Storage, it implies that most challenges in data security relate to confidentiality rather than integrity or authenticity.

Category - Software Vulnerabilities This category relates to security weaknesses present in a software system that can be exploited by an attacker. This involves the explicit flaws in the code or security features.

– **Network Exploits:** Network exploits and vulnerabilities, such as SQL Injection, XSS, CSRF and DDoS, as well as methods to prevent them, such as input validation. For example, "*How can prepared statements protect from SQL injection attacks?*" *(SO, 8263371)*. As these attacks are often targeted at web applications, this topic is noticeably most prominent in web-oriented languages, i.e., PHP, JavaScript, Ruby and TypeScript. However, challenges are most frequent for Python on Stack Overflow, with 18% Topic Share. This is because of Python's *Django* framework enforcing CSRF protection, leading to many Stack Overflow questions. For example, "*When I try to send POST request, Django requires CSRF token. How can I send POST request without any problems?*" *(SO, 23743273)*.

– **Mitigation:** Vulnerability reports, scanning and patches. This topic does not fall under other sub-categories as the discussion is often not explicit to any particular vulnerability. For instance, the GitHub issue *"chore: fix vulnerability" (GH, strongloop/loopback-next/pull/2880)* does not provide details about the actual vulnerability it fixes. Like Network Exploits, this topic is evident for web-oriented languages: JavaScript, Ruby and TypeScript.

– **Memory Management:** Discussion relating to the management and mitigation of memory related issues. The dominant challenges in this topic relate to stack/heap memory and overflows, e.g., "*buffer overflow exploits - Why is the shellcode put before the return address*" *(SO, 16789241)*. Individual topics can also relate to dynamic memory linking methods, such as pointers, but this discussion is only prominent in C and C# as these mechanisms are often uncommon in other languages. GitHub issues for this topic commonly relate to memory leaks, e.g., "*Fixes memory leaks in IoT*" *(GH, aws-amplify/aws-sdk-ios/pull/1175)*. This topic is only dominant for C/C++, and discussion is noticeably lacking for other languages. This is potentially expected, as these languages are memory and type unsafe, but these vulnerabilities can still manifest in other programming languages, such as PHP (Cifuentes and Bierman 2019).

– **Race Condition:** Secure and safe management of access to shared resources by multithreaded or parallel processes, e.g., "*Can Multiple Threads Read The Same Class Member Variable?*" *(SO, 59191339)*. This is the least prominent sub-category, and is only evident for Java, C#, and C/C++.

Category - Secure Development Secure development refers to the practices and tasks involved with producing secure software. These discussions do not hold any explicit relation to the other categories. For example, debugging a security feature, or configuring a secure framework. This category has the highest overall topic share value, as the major focus of the collected dataset is secure development. As a result, discussion is prominent for all languages in our dataset.

– **Implementation:** Implementation or requirements planning of security elements, e.g., "*Submit button is not getting enabled after checking all the fields*" *(SO, 34253518)* or "*Add best practice encryption requirements.*" *(GH, coreinfrastructure/best-practices-badge/pull/3)*.

– **Testing:** Testing and debugging for general security defects or bugs, e.g., "*initializationError FAILED java.lang.SecurityException*" *(SO, 29508622)*.

– **Libraries/Configuration:** Integration and configuration of security-related libraries, frameworks, plugins or addons; e.g., "*spring security 3.2 java configuration*" *(SO, 26944600)*.
– **GitHub/Reports:** General GitHub development activities and reporting of software issues and bugs, such as GitHub issues and pull requests, or bug reports. For GitHub, this topic is heavily influenced by GitHub bots, e.g., *googlebot*[16] which requests the user to sign a license agreement. This topic is not very pervasive on Stack Overflow as expected, but still appears for Shell due to the frequent usage of Git via the command line, e.g., "*How to switch from first github account to second?*" *(SO, 37782617)*.

The existing security state of the art also provides taxonomies for security theory and knowledge. However, none of these taxonomies are empirically constructed through development challenges or mapped to programming languages like ours. We position our findings with respect to the state of the art by comparing our identified taxonomy to two existing security taxonomies. The Cybersecurity Body Of Knowledge (CyBOK) (Martin et al. 2019), is a mapping and collection of cybersecurity knowledge published in textbooks, academic research articles and technical reports. It identifies 20 knowledgebases for five key domains. Additionally, the Common Weakness Enumeration (CWE) presents a taxonomy of security weaknesses for software security (CWE 699).[17] We compare our taxonomy to these existing security taxonomies in Fig. 9.

As our taxonomy is constructed empirically from development rather than security theory, we notice several differences to the existing taxonomies. CyBOK is oriented towards education, and as a result covers security aspects very broadly. Only two of the domains, *System Security* and *Software Platform Security*, relate to the development challenges that we identify. Furthermore, the sub-categories of the relevant chapters do not match the granularity of our analysis and are often topics that are quite specific to theoretical aspects, such as *Cryptographic Security Models*. Additionally, the existing state of the art is not provided from a programming language perspective, which can make application difficult for practitioners, managers, educators, and researchers who are concerned about this view.

The CWE taxonomy is primarily oriented towards vulnerabilities and exploits, which only covers a small portion of our dataset. Challenges often relate to development and implementation of security features rather than discussion of explicit flaws. Some of the more simplistic vulnerabilities are not covered in our taxonomies, such as Numeric, String and Type Errors. Although these vulnerabilities may still receive some discussion, they do not experience enough discussion in developer communities to manifest as significant challenges through our topic modelling process.

Although, CWE entries are substantiated using references to formal public reports and vulnerability databases, these references are not systematic nor evaluated with respect to publicly available security discussions and sources, like our taxonomy is. Additionally, CWE provides the applicable platforms/programming languages for some CWE types, but this information is considered far from complete. For instance, 75% of CWE types which list Applicable Platforms (423 out of 566) are categorized as *Language-Independent*, which may be theoretically true, but through our study we find the prevalence and nature of the security challenges to vary across programming language. Furthermore, 63% (362 out of

---

[16]https://developers.google.com/open-source/github/accounts#googlebot
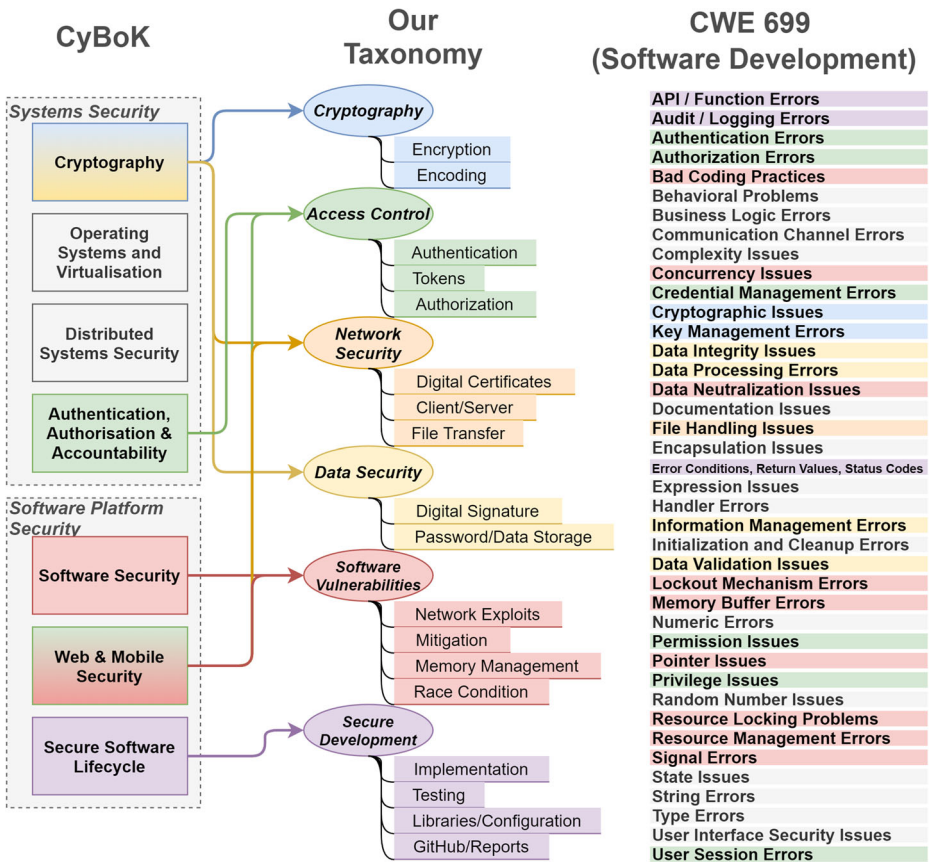[17]https://cwe.mitre.org/data/definitions/699.html

Fig. 9 The comparison of our discovered security taxonomy to the security state of the art

571) of the language examples on CWE are for Java and C, which does not provide a wide view of different programming languages.

> **RQ3 Summary:** We identify 18 topics and 6 topic categories: Cryptography, Access Control, Network Security, Data Security, Software Vulnerabilities, and Secure Development. Secure Development is the overall most prominent topic category, followed by Access Control. Different languages experience different topic distributions.

4.4 RQ4: How do Security Discussion Topics Change Over Time for Different Programming Languages?

We examine the topic impact values of each topic category for each language on Stack Overflow and GitHub for the time span of January 2009 to May 2020. Using the Mann-

Kendall Trend Test (Hussain and Mahmud 2019) with a 99% significance level, we identify each category as either increasing, decreasing or no-trend. The results can be seen in Table 8.

Topic Impact appears to be predominantly increasing for most topic categories, especially amongst languages on GitHub and topics. Secure Development in particular exhibits a steadily increasing trend across nearly all languages and sources.

All languages exhibit an overall increasing trend in security challenges, particularly for Secure Development. This implies a general increase in security discussion and challenges over the past decade. PHP is the only language which has a higher number of decreasing topic trends across both sources.

As questions, issues and discussions usually rise in response to problems, an increasing Topic Impact value may imply a rise in challenges. However, an increase in discussion also represents increased awareness and a larger knowledge-base, especially for Stack Overflow.

The Access Control topic category is overall decreasing on GitHub, which could be interpreted as errors in Access Control are not occurring as commonly. PHP, Python and Go have an increasing trend for Access Control on Stack Overflow, despite the decreasing trend on GitHub. Access Control challenges are still increasing for JavaScript, Swift and TypeScript on both sites.

For Stack Overflow, there are several decreasing trends for Network Security and Data Security. This implies a growing lack of prevalence on these topics for the relevant languages. These decreasing trends are for the most popular languages of Stack Overflow developers.[18] Thus, these languages may be moving to other areas of focus due to their popularity. For Network Security, none of these trends are reflected on GitHub, which implies they are still important factors of development. However for Data Security, this topic is not represented commonly on GitHub.

These decreasing trends may be further supported by the sites attempted avoidance of duplicate questions. Security questions and discussions that are posted earlier are typically not re-posted, as users have the ability to visit the original post.

We examine the five most significant positive and negative trends for both Stack Overflow and GitHub in Fig. 10, as calculated by the Theil-Sen's Slope estimation value (Hussain and Mahmud 2019). We use a rolling average of window size 3 to help smooth the noise in these trend graphs.

The major increases in both Stack Overflow and GitHub language security appear to be caused by the inception of new languages. The major spikes in the top left graph of Fig. 10 align to the date that these languages first commonly appeared; Go: 2009, TypeScript: 2012, Swift: 2014. As such, the trends for these graphs are noisy, but we can still observe an initial surge of interest followed by a steady increase in discussion/challenges for the various security topics of these languages, after their original inception.

This trend may be caused by general inexperience with these newer languages or lacking documentation. For example, *"I am new to typescript. I am able to integrate auth0 using javascript because auth0 providing a sample for that but there is no sample app available for vue with typescript." (SO, 59135589)*. This user admits that they are a novice to TypeScript and are struggling to find support for this language.

---

[18]https://insights.stackoverflow.com/survey/2020

Table 8 Overall topic impact trend from January 2009 to May 2020

Programming Language	Topic Category					
	Cryptography	Access Control	Network Security	Data Security	Software Vulnerabilities	Secure Development
Java						
PHP						
JavaScript						
C#						
Python						
Shell						
C						
Ruby						
Objective-C						
Swift						
PowerShell						
Perl						
Go						
TypeScript						

The symbol on the left indicates the Stack Overflow trend and the symbol on the right indicates GitHub. An up arrow represents an increasing trend, a down arrow represents a decreasing trend, a dash represents no trend, and a cross represents no topics for that category.
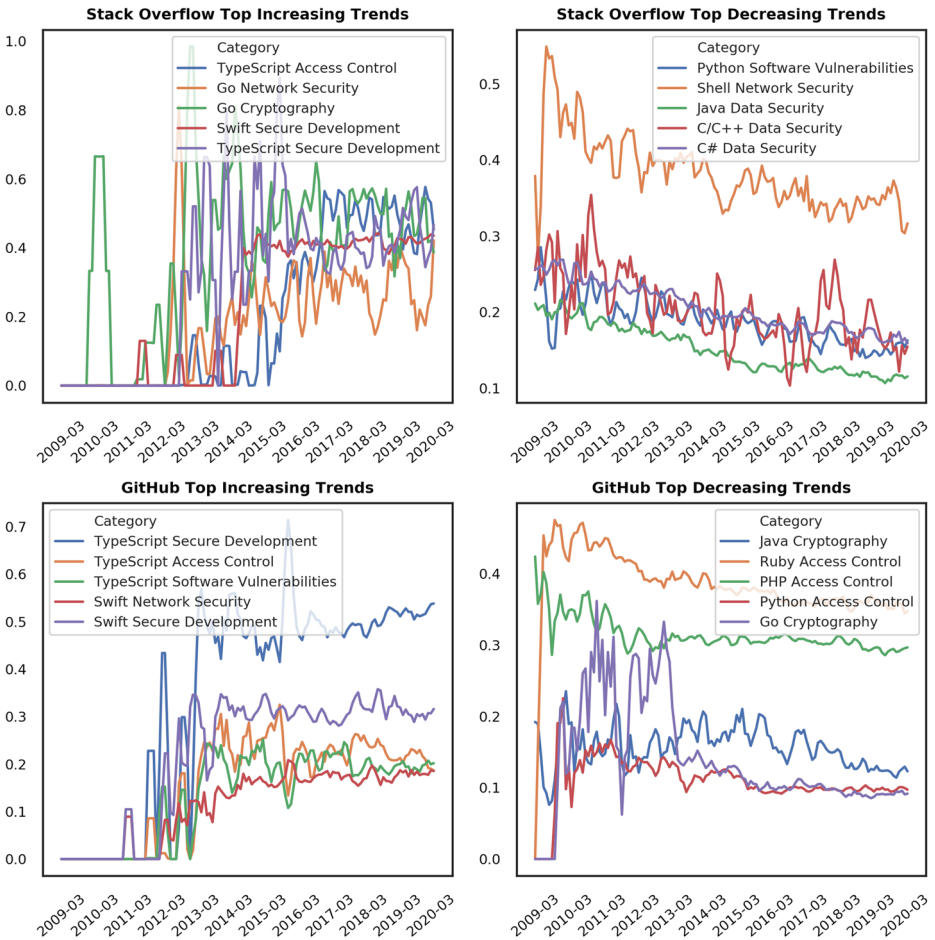
Fig. 10 The 5 most significant increasing and decreasing topic category trends for Stack Overflow and GitHub

> **RQ4 Summary:** The most popular languages experience decreasing challenges in the areas of: Access Control (GitHub), Network Security (Stack Overflow), and Data Security (Stack Overflow). However, challenges are overall increasing for most languages, except for PHP. The most significant trends are for new languages, which see a steady rise in challenges after their introduction.

4.5 RQ5: What are the Characteristics in Terms of Popularity and Difficulty of Different Programming Languages and Their Identified Security Challenges?

The popularity and difficulty metrics provide insights into the characteristics of the identified and analyzed security challenges. Through these insights we can better inform developers of the nature of these challenges; their ability to be resolved and handled. The

popularity metric gives an indication of the interest and involvement of users for these challenges, which suggests how readily these challenges will be solved by online communities. The difficulty metric gives an indication of the complexity and hardness of the challenges; how much knowledge and labour is required to overcome the identified issues. The combination of these metrics gives an indication of how significant the challenges are when they occur. High popularity and low difficulty challenges are preferable as they would be expected to be resolved relative quickly and easily.

The overall normalized popularity and difficulty for security of the languages for each source is presented in Fig. 11. It is expected that developers would favor languages which are both popular and not difficult; as it implies that there is an abundant amount of interest and resources to solve potential issues, and that the encountered problems are not as challenging. For instance, Meyerovich and Rabkin (2013) find language popularity and learning cost to be important factors for language adoption.

We observe that there is no significant correlation between the Stack Overflow and GitHub metric values, using Kendall's Rank Correlation Coefficient (Knight 1966). Similarly, there is no significant correlation between the Popularity and Difficulty values, which implies that the most popular languages and topics for security are not necessarily the least
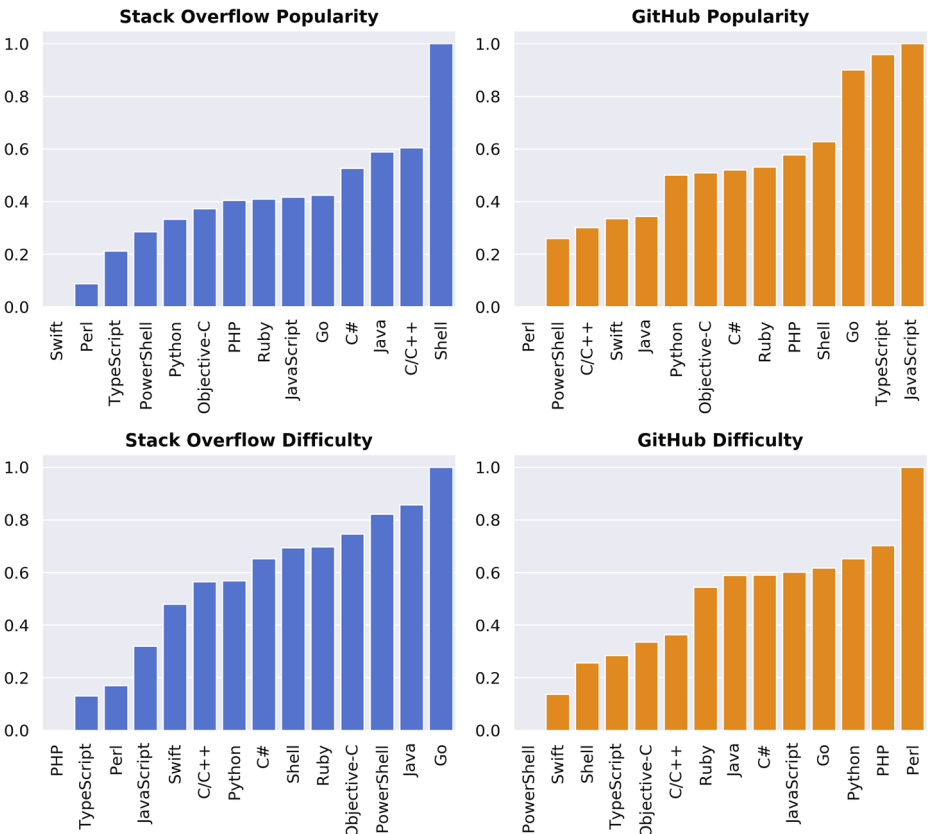
Fig. 11 The normalized popularity and difficulty values of each language for Stack Overflow and GitHub

difficult and vice versa. Figure 12 displays the normalized popularity against difficulty for each language and source.

On Stack Overflow, security related posts for Shell are significantly the most popular. Upon closer inspection of the individual metrics, this is because Shell related security posts receive significantly more views from the community. We find that high view posts for Shell commonly relate to SSH Keys, an authentication method for an ecrypted connection between systems. For instance, *"Automatically enter SSH password with script"* or *"Calculate RSA key fingerprint"*. The popularity of these posts is likely due to the diverse usage of SSH keys regardless of the application context. C/C++, Java and C# are also popular on Stack Overflow.

The most popular languages on GitHub for security are JavaScript, TypeScript and Go. The popularity of JavaScript and TypeScript on GitHub implies that security is a very popular topic for web development, receiving active interest from the community. Security
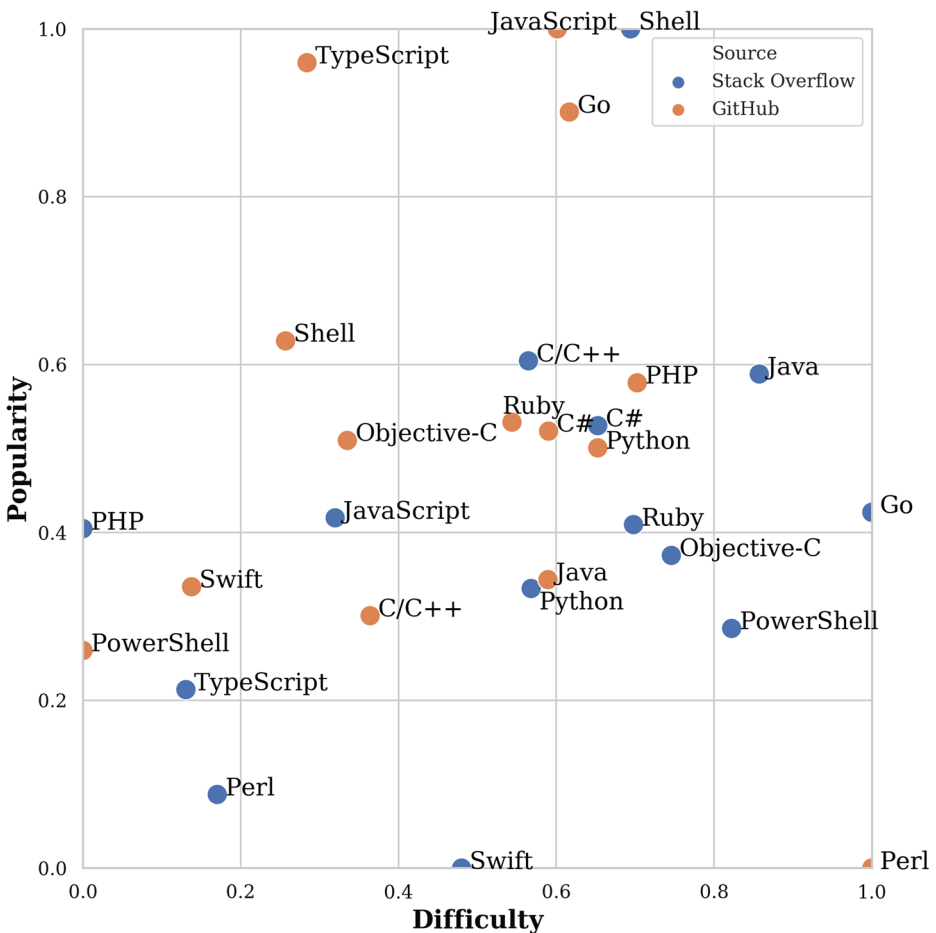
Fig. 12 The normalized popularity against the normalized difficulty values of each language. Sources are normalized separately

related posts for web development languages (i.e., PHP, TypeScript, JavaScript) are also the least difficult on Stack Overflow.

Perl appears to be an undesirable language amongst the community for security. Posts receive very little attention (Popularity) from both the Stack Overflow and GitHub community. It also has the most difficult to resolve issues on GitHub.

We further summarize the categories for which each language is considered most and least Popular/Difficult in Table 9. The most popular categories for a language receive the most user attention, and the least difficult categories are resolved the fastest and most easily. Hence, from Table 9 practitioners can infer which security aspects of a language receive the most and least community support. Languages which have an inverse ranking for popularity and difficulty are highlighted in bold.

Table 9 The most and least Popular and Difficult topic categories for each language

Metric	Cryptography	Access Control	Network Security	Data Security	Software Vulnerabilities	Secure Development
Stack Overflow						
Most Popular	***Shell***, Objective-C, ***Perl***	–	PowerShell, ***Go***	Java, PHP, C#, Ruby, Swift	Python, ***C***, ***TypeScript***	JavaScript
Least Difficult	Python, ***Shell***, ***Perl***	Objective-C, PowerShell	***Go***	JavaScript	Java, PHP, C#, ***C***, Ruby, ***TypeScript***	Swift
Least Popular	Ruby	Java, PHP, Python, Objective-C, PowerShell, Perl, ***Go***, ***TypeScript***	***Shell***, ***C***	JavaScript	C#	Swift
Most Difficult	PHP, JavaScript	***Go***, ***TypeScript***	Python, ***Shell***, ***C***, Ruby, Objective-C, PowerShell	Swift	–	Java, C#, Perl
GitHub						
Most Popular	C#	–	JavaScript, Python, Ruby, ***TypeScript***	***PHP***, Shell, C, ***Objective-C***, Swift, Perl	–	Java, ***PowerShell***, Go
Least Difficult	Swift	C#	***TypeScript***	***PHP***, Python, ***Objective-C***	Java, JavaScript, C, Go	Shell, Ruby, ***PowerShell***, Perl
Least Popular	JavaScript, Python	Ruby	Shell, Swift	–	Java, PHP, C, ***Objective-C***, Go, TypeScript	C#, ***PowerShell***, Perl
Most Difficult	C#, C	PHP, Swift, Go, TypeScript	JavaScript, Ruby	Shell, Perl	***Objective-C***	Java, Python, ***PowerShell***

The individual values for the metrics that comprise the Popularity and Difficulty Score of each topic for each language can be seen in our online appendix.[19]

> **RQ5 Summary:** Popularity and Difficulty differs heavily for languages across Stack Overflow and GitHub. Web-Oriented languages are the most popular on GitHub, whereas they are the least difficult on Stack Overflow. Similarly, different languages exhibit different rankings for individual topic categories.

4.6 RQ6: What is the Level of Security Expertise of the Users who Answer Security Related Discussions for Different Programming Languages?

The level of security expertise gives an indication of a user's knowledge and experience for security related topics. For Stack Overflow users, a high expertise value implies a wealth of knowledge, either theoretical or practical, and the ability to convey this knowledge well to other users. For GitHub users, a high expertise value indicates a lot of experience in software development and practices for high quality projects. Through this we can infer whether there are enough experts to resolve the security challenges that a developer may face for a particular programming language, and the quality of the provided answers. A language with high expertise users is considered preferable, as it would be expected that answers and hence resolutions are of higher quality. This research question helps us better understand how the challenges are resolved. The normalized mean security expertise values for all users who answer security related questions in our dataset is presented in Fig. 13.

Similar to RQ5, we again observe a disconnect between Stack Overflow and GitHub. For GitHub, the languages with the highest security expertise are languages commonly used for mobile development (i.e., Objective-C, Swift, Java, JavaScript, C#). However, on Stack Overflow the expertise of users for these languages is relatively low. This might suggest that their is an abundance of experienced secure mobile developers, but their ability to detail knowledge is lacking.

For Stack Overflow, languages with a lower level of abstraction (Shell, C/C++) have relatively higher expertise values. These languages are typically considered both type and memory unsafe which implies a greater level of security knowledge is required to avoid potential vulnerabilities. Hence, it is a promising sign that the average security expertise of users' answers is relatively high. These languages also rank highly on GitHub, but still have a much lower expertise value than the aforementioned mobile-related languages. Go also ranks highly for Stack Overflow. Whilst this language is syntactically similar to C and C++, it has added memory safety and garbage collection capabilities.

Perl ranks as the language with the highest average security expertise for Stack Overflow users, which may explain its low difficulty values amongst this community in RQ5. However, the security expertise of Perl users on GitHub is relatively low, which is potentially expected as Perl is the least popular and most difficult language for GitHub.

Except for JavaScript on GitHub, web-oriented languages (i.e., PHP, Ruby, TypeScript, JavaScript) have a relatively low average user security expertise on both sources. This implies that the answers given to challenges for these languages are fairly simplistic.

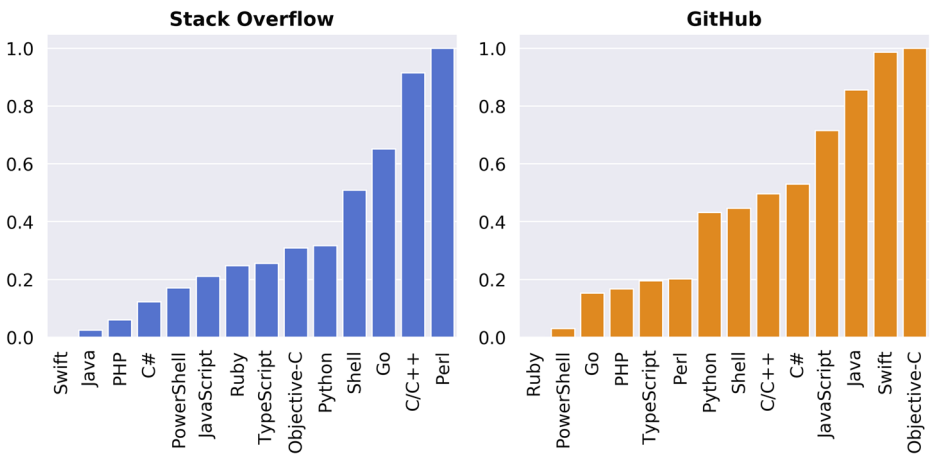[19] https://github.com/RolandCroft/Language-Security-Challenges/tree/master/Document_Metrics

Fig. 13 The normalized mean security expertise values of each language for Stack Overflow and GitHub

> **RQ6 Summary:** Language security expertise heavily differs for users of Stack Overflow and GitHub. Mobile development oriented languages have the highest expertise users on GitHub, whereas languages with lower levels of abstraction have the highest expertise users on Stack Overflow.

5 Discussion

The goal of this study is to empirically investigate and compare developer security challenges across different languages. To achieve this, we investigated 6 research questions to identify the different programming language security consideration, discussions and support.

We observe that security related topics are relatively infrequent, only forming an average of 1.23% of all discussion for programming languages. In a recent developer survey by Venson et al. (2019), it was estimated that security accounts for an average of 20% of the development effort for most projects, which is considerably higher than our observed frequency. This suggests that there needs to be more consideration for security aspects, especially amongst programming languages. Shell and PHP exhibit high rates of discussion (2.6% and 2.1%), approximately double the average rate, which indicates that security is properly appreciated for these languages. However, discussion and issues are almost non-existent for several languages, such as R, Coq, Julia, Fortran and MATLAB, which have an average rate of 0.1%. This shows a lack of prioritisation or awareness of the security needs in development using these languages, which should be improved.

Across the two sources, we find that the purpose of these discussions is most commonly caused by *Errors* which users are trying to resolve. This indicates that secure development

in most programming languages is difficult or error-prone. This is inverse to general development discussions (Beyer et al. 2020) which observe *Errors* to be one of the less common question categories on Stack Overflow. We also observe C/C++ discussions to have a significantly higher focus on *Review*, indicating a much higher examination of security for these languages. This aligns with the findings reported by Bhattacharya and Neamtiu (2011) that code quality is higher for C/C++ software repositories. However, it has been found that there are a large number of software vulnerabilities in crowd-sourced code examples for C/C++ (Verdi et al. 2020; Zhang et al. 2021), which implies that security is not always the main consideration for this language.

The majority of programming languages have Secure Development as the most dominant topic category. This shows that programming language specific challenges are more oriented towards development tasks, such as implementation, testing, and configuration; there is less regard for theory and concepts. These findings suggest that developers still need a better understanding and education of the security concepts that they are required to implement. However, we identify that the challenges of our created security taxonomy are prevalent across many different programming languages, which indicates that developers encounter a wide array of problems regardless of technology, and urges that more solutions need to be provided.

From our analysis in Section 4, we identify that the security challenges and their characteristics are often oriented towards the task that the language is commonly utilized for. We focus on Web and Mobile development as many programming languages are explicitly applied for these domains (Sebesta 2012), such as JavaScript and TypeScript (Web), or Objective-C and Swift (Mobile). For general multi-purpose programming languages, such as Python and Go, it is more difficult to observe the trends.

Web-Oriented languages have the most prevalent challenges in Access Control and Network Security, whereas Mobile-Oriented languages have the most prominent challenges in Cryptography, Access Control and Data Security. Another programming language trend we observed are for languages with a lower-level of abstraction. That is, languages which have strong coupling or control of the underlying operating system and architecture (i.e., C/C++, Shell). We observe that their posts on Stack Overflow receive the most popularity, as well as the most attention from expert users.

We can speculate the reason for these trends through the application domain. For instance, if a programming language, such as JavaScript, is most commonly used for Web Development, then we can expect the major security issues to involve Access Control and Network Security. Web Development is a common task on GitHub, so we can also expect that these issues would receive high engagement (popularity) from the GitHub community.

5.1 Implications

For Developers We firstly help practitioners to understand the impact that the choice of programming language can impose on secure development. Our findings can help developers to gain security awareness and insight into their selected programming languages. We identify what the prevalent security challenges of each language are, detail how common and readily the challenges are solved, how difficult the challenges are, and whether there is enough expertise in the community to help solve them. Through these findings, developers can identify what areas they should be better informed about and what areas they are able to get help in. We provide preliminary recommendations based on the most significant observations of our study.

Firstly, developers should avoid security critical programming whilst using languages primarily designed for scientific programming (i.e., R, Julia and MATLAB). We observe in RQ1 that users of these languages have the lowest consideration for security, which implies that they are not designed for this task and receiving community support will be challenging.

Developers should be more aware of the relation of security vulnerabilities to their specific development context, and how to defend against them. From our constructed security taxonomy, we found "Software Vulnerabilities" to be the least prominently discussed security topic for programming languages. Aside from "Network Exploit" discussions for web-oriented languages, and discussion of "Memory Management" for C/C++, these topics are largely non-existent for programming language focused discussion, which aligns with the findings of Zahedi et al. (2018). Whilst most vulnerabilities are language-independent, it is still critical that developers have specific implementation knowledge of how to avoid them.

In RQ5 and RQ6, we observe that security implementation for TypeScript and JavaScript is relatively straightforward amongst the analysed communities. Challenges are resolved relatively quickly (low difficulty values on Stack Overflow), and issues receive a lot of attention (high popularity on GitHub). Despite the popularity and low difficulty, security users for these two languages are also generally of lower expertise than other languages. This could imply that sufficient security knowledge can be obtained relatively easily for these languages, as users are able to quickly share security knowledge and resolve issues, despite having lower average experience (as defined by GitHub expertise in Section 3.5).

We also identify that the concerns and interests of developers vary between Stack Overflow and GitHub. For example, for RQ5 and RQ6 we find that Web-oriented languages have high popularity security issues on Stack Overflow but are low difficulty on GitHub, and that users have more security expertise for Mobile-oriented languages on GitHub than on Stack Overflow. These findings can help developers to find appropriate answers and discussions for their problems. We elaborate this further in Section 5.2.

Additionally, our findings help motivate the creation of better security documentation and APIs for languages in the categories which face challenges. This effort will allow for easier secure use of the language and help reduce the faced challenges. For RQ2, we observe *API Usage* to be the main reason for programming language security challenges on Stack Overflow. Furthermore, we find C/C++ to have a significantly higher proportion of discussions requesting for code review and improvement, implying that these users are often incapable of ensuring security on their own. Whilst users of C/C++ seem to be aware of the need for security, experienced developers should help ensure that appropriate resources and knowledge is available.

We especially observe the need for better security support and resources for new programming languages, as we find users have significant security challenges after the initial release. This aligns with the findings of Chakraborty et al. (2021), who identify that adequate community resources are initially lacking for new programming languages.

For Managers Although developers are responsible for their own personal programming language preferences, which may influence a project manager's decision, project managers ultimately have final say in programming language selection for a project (Naiditch 1999). Our study helps raise awareness of the importance of programming language selection and evaluation, promoting them to reserve project resources and time for these tasks. Through our comparison of security challenges for different languages we can help managers in their decision making for choosing what programming languages to use when starting a new project. Particularly through the characteristics of the challenges analysed in RQ5 and RQ6,

we can obtain an indication of what languages are better suited to different categories of development. However, we acknowledge that not all languages can be used for all projects, and language is sometimes unable to be chosen due to external factors, such as legacy code. Thus, our findings and the identified challenges can also help guide necessary developer training decisions; either for new security development tasks, or for new programming language use. Additionally, educators can use the knowledge that we have synthesized from publicly available data as a potential guide for important topics in security education or a check-list for required security knowledge.

For Researchers The challenges we have identified can help direct and motivate researchers to investigate potential solutions. The identified taxonomy of security issues can help direct and align this security language research. Furthermore, our proposed method of sampling, cleaning and analysing the data can be used by future researchers to identify the challenges and knowledge of other domains. The findings of our study also suggest that programming language should continue to be an important consideration in future security research, as we have observed the challenges and their nature to differ for different programming languages. Language consideration could also be potentially strengthened in several research areas. For instance, vulnerability prediction models often do not consider the characteristics or features of the languages that they target (Yang et al. 2017; Croft et al. 2021).

Additionally, we arouse the attention of language designers for the impacts that programming language can have on secure development. Our findings help identify the major security issues and considerations of developers, which can assist designers in improving the supportive capabilities of security features in programming languages.

5.2 Difference in Discussion between Stack Overflow and GitHub

Interestingly, for RQ3-6 we observe a consistent disconnect between the security characteristics of programming languages on Stack Overflow and GitHub. For instance, PHP and Perl are some of the most difficult languages for security on GitHub but the least difficult on Stack Overflow. Furthermore, the highest popularity security issues on GitHub are for TypeScript and JavaScript, whereas the issues for these languages are relatively unpopular on Stack Overflow. This separation is consistent with the analysis reported by Han et al. (2020), who also observe a disparity between Stack Overflow and GitHub.

Through this disconnect, we can derive recommendations for where users should go for assistance with security related challenges of a particular language. Although users do not traditionally visit GitHub for answers to programming problems, GitHub issues of open source projects are still a viable resource for developers to query project specific questions; questions have even been observed to be the third most common issue type Cabot et al. (2015). Hence, for languages more oriented towards software and app development (e.g., JavaScript, TypeScript, Swift, Objective-C), users should seek answers on GitHub, as these challenges are more popular, less difficult and answered by more experienced users on this platform. GitHub also exhibits a higher general rate of discussion for Secure Development. For other languages, particularly general-purpose languages (e.g., Java, Python) or languages with lower abstraction (e.g., C/C++, Shell), users should seek help from the Stack Overflow community.

We conjecture that work should be done in connecting these two sources, as knowledge gained from one source may not adequately prepare users. This is showcased through user expertise, where we observe that Stack Overflow users are relatively inexperienced with

Mobile-Oriented languages, whereas the users of these languages on GitHub exhibit the most experience.

5.3 Vulnerabilities for Different Programming Languages

Another approach for determining the security of a programming language is to examine how prone it is to vulnerabilities. By adopting a similar approach to Decan et al. (2018) and Gkortzis et al. (2018), we can determine an indication of language vulnerabilities in their packages and projects, via the vulnerability advisories *Snyk.io* and *National Vulnerability Database (NVD)*, respectively. Through this method we identify that projects written in C and PHP, and packages written for Java, have the highest number of reported vulnerabilities. However, few assertions can be made from this information. It does not take into account the prevalence of the language; C is one of the oldest popular programming languages, and Java is one of the most commonly used languages. Similarly, vulnerabilities in certain languages are more well known and easier to detect (Shahriar and Zulkernine 2012), and the number of annual reported vulnerabilities is generally increasing for all languages.[20]

Furthermore, the representation and reliability of NVD data has also been questioned by several previous works (Massacci and Nguyen 2010; Nguyen and Massacci 2013). NVD's nature of only reporting observed and documented vulnerabilities has been shown to potentially obscure and misconstrue the true vulnerabilities of a project (Massacci and Nguyen 2010). Nguyen and Massacci (2013) also investigated the reported vulnerabilities in Google Chrome and found several errors. It is further observed that developers themselves do not necessarily perform consistent reporting of vulnerabilities. For instance, this GitHub issue[21] discusses whether certain vulnerabilities should be reported to a vulnerability database. Thus, vulnerability databases cannot be considered as a definitive view of software vulnerabilities.

For the purposes of this study, we do not explicitly consider vulnerabilities as it is difficult to draw conclusions from this data alone; vulnerabilities are not directly equivalent to secure development challenges. Vulnerabilities only indicate the specific exploits and weaknesses in a system. They also rely on proper identification, documentation and reporting. It might be expected that some developers are unwilling to document or discuss vulnerabilities in their software publicly. Instead we try to examine the complete picture by also investigating secure implementation, coding, defenses, knowledge and theory.

5.4 Reliability of Crowd-sourced Knowledge

Despite the existence of crowd-sourced security knowledge in our examined data sources, its reliability has been a source of contention in past research. There is evidence that code snippets on Stack Overflow contain security flaws (Acar et al. 2016; Fischer et al. 2017), and a recent study into the security answer posts on Stack Overflow found almost 45% to be insecure (Chen et al. 2019). However, approximately 70% of questions on Stack Overflow receive an accepted answer,[22] which implies that discussion and knowledge-sharing is often successful and informative for the original poster in the least.

---

[20]https://www.whitesourcesoftware.com/most-secure-programming-languages/
[21]https://github.com/bcit-ci/CodeIgniter/issues/4020
[22]https://stackoverflow.com/questions?tab=Unanswered

The success and validity of security discussion in GitHub issues also lacks confirmation. We find that as of July 2020, 88% of the issues in our GitHub dataset are closed. If we similarly consider closed GitHub issues as a discussion with a successful resolution, then this also implies that the majority of GitHub issue discussions achieve a sufficient level of conclusion.

6 Threats to Validity

Internal Validity We use topic modelling to automatically cluster documents, and identify topics of discussion. Due to the size of our dataset, we are required to use the semi-automated approach of topic modelling for analysis. However, the topics and clusters produced by topic modelling are often volatile (Mantyla et al. 2018), and the interpretation of the produced topics is often subjective. We note that the topics we identify in Section 4.3. are not exhaustive or fully representative of the security challenges, but are the most prevalent in the text data. We use manual validation to help confirm the accuracy of the topics, and cross-check our labels to help confirm their validity. Furthermore, although the metrics outlined in Section 3.5 are able to give an indication of the characteristics of the data, we acknowledge that they are not perfect.

The rigidness of the produced topics also limits the versatility of our identified security categories and challenges. Documents can only be assigned to the available topics. If a security topic is not reported for a language, it often does not mean that it does not exist, but that is obscured by a more dominant topic.

Due to the nature of our empirical study, we acknowledge that there are threats which affect our conclusions regarding the correlation between use of programming language and security challenges. Our findings may be influenced by several confounding factors, such as expertise of the users behind the discussions. However, we observe several potentially interesting differences across the security challenges, which we report. These can be used as a starting point for future research to help confirm our findings.

Similarly, the large size scale of the analysis prevents us from providing specific insight into the experience or engagement within these programming language communities, and we hence use popularity, difficulty and expertise metrics defined in Section 3.5 as proxies for our analysis. However, we are still able to provide important insight into the prevalent security activity within these different communities.

External Validity We conduct an extremely large-scale study to help improve the generalizability of our findings by investigating two of the most prominent sources for developer community support; Stack Overflow and GitHub. However, Stack Overflow and GitHub may not be perfectly representative of all developers. We also discuss in Section 5.2 and Section 5.3 that our findings are likely to change across different datasets. Due to the nature of our study and methods, our findings can only be considered reliable for our selected datasets.

For GitHub, our method of data collection has further flaws due to our repository sampling method. The data of each language may be biased towards the repositories that were sampled, which may obscure the true trends of the underlying language. However, this assertion can be correct for several other similar empirical studies.

As the data sources from which we extract the data from are subject to change, our study is not perfectly reproducible. For both Stack Overflow and GitHub, posts can be added, removed, deleted, changed or updated, and thus someone using the same method as us

would likely extract a slightly different dataset. Similarly, the posts referenced in this paper for discussion may not always be available if they are deleted or removed.

Construct Validity Keyword and tag matching are not perfect methods for data extraction, as they rely on a manually constructed keyword set and correct use of the keywords by the users of the dataset (e.g., correct spelling, appropriate post tagging). However, we manually review several samples of posts in our dataset to help confirm their relevance to the topic our study, and thus help confirm the validity of our approach.

There is also the threat to validity regarding the implementation of the study. We independently reviewed our scripts and code to check for correctness and reduce this potential threat. We have also made our data and results publicly available for reproducibility.

7 Conclusion and Future Work

We have presented a large-scale comparative analysis of the security challenges encountered during development using different programming languages. To conduct this analysis, we examined approximately 280,000 publicly accessible security-related developer discussions from Stack Overflow and GitHub for 15 popular programming languages. We also investigated the nature of these challenges and their ability to be resolved to help provide recommendations to developers.

Our findings show that languages exhibit varying rates of security discussion, categories of security issues, and characteristics in which these security issues are handled by the community. Language security trends emerge for languages with similarly oriented domains, particularly web-oriented languages, mobile-oriented languages, and lower-level abstraction languages. The observed security challenges are handled better by different languages for different categories and different sources. For example, it is suggested that TypeScript is well suited for "Network Security" and avoiding "Network Exploits", as we observe the discussed issues to have high popularity and low difficulty amongst the community.

Our study is expected to motivate the need for practitioners to consider and evaluate the influence that programming languages might impose on secure software implementation. Our findings can help them understand and navigate the secure development landscape for different languages. Additionally, our identified taxonomy of security challenges have the potential to guide and increase the productivity of developer training, or secure software engineering research.

For future work we plan to investigate whether the difference in challenges for different programming languages is also reflected in the code and software. To achieve this we intend to analyse the explicit security situations of different open-source repositories in more detail, by examining the vulnerabilities, software metrics, and code quality. We particularly aim to investigate Software Vulnerability related issues, as we noticed discussion for these topics to be lacking for most programming languages. Additionally, we intend to extend our study to also consider technology stacks (i.e., libraries and frameworks). We also aim to conduct more fine-grained analysis into users' expertise and engagement through more extensive qualitative analysis or a user study. This additional analysis should help support our findings and provide more specific insights. Furthermore, our analysis in Fig. 9 reveals insights into how open source security challenges align and intersect with more formal security taxonomies from CyBoK and CWE. We aim to investigate in future how these software and security communities can be brought together.

Data Availability All data has been made available via an online appendix.

Code Availability Not applicable.

Declarations

Conflict of Interests Not applicable.

References

Acar Y, Backes M, Fahl S, Kim D, Mazurek ML, Stransky C (2016) You get where you're looking for: The impact of information sources on code security. In: 2016 IEEE symposium on security and privacy (SP). pp 289–305

Agrawal A, Fu W, Menzies T (2018) What is wrong with topic modeling? and how to fix it using search-based software engineering. Inf Softw Technol 98:74–88

Ahmad A, Feng C, Ge S, Yousif A (2017) A survey on mining stack overflow: Question and answering (q&a) community. Data Technol Appl 52:190–247

Ahmed S, Bagherzadeh M (2018) What do concurrency developers ask about? a large-scale study using stack overflow. In: Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement. pp 1–10

Allamanis M, Sutton C (2013) Why, when, and what: analyzing stack overflow questions by topic, type, and code. In: 2013 10th working conference on mining software repositories (MSR). IEEE, pp 53–56

Bagherzadeh M, Khatchadourian R (2019) Going big: a large-scale study on what big data developers ask. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. pp 432–442

Bajaj K, Pattabiraman K, Mesbah A (2014) Mining questions asked by web developers. In: Proceedings of the 11th working conference on mining software repositories. pp 112–121

Bangash AA, Sahar H, Chowdhury S, Wong AW, Hindle A, Ali K (2019) What do developers know about machine learning: a study of ml discussions on stackoverflow, IEEE

Barnum S, McGraw G (2005) Knowledge for software security. IEEE Secur Privacy 3(2):74–78

Barua A, Thomas SW, Hassan AE (2014) What are developers talking about? an analysis of topics and trends in stack overflow. Empir Softw Eng 19(3):619–654

Bayati S, Heidary M (2016) Information security in software engineering, analysis of developers communications about security in social q&a website. In: Pacific-Asia workshop on intelligence and security informatics. Springer, pp 193–202

Beyer S, Macho C, Di Penta M, Pinzger M (2020) What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. Empir Softw Eng 25(3):2258–2301

Bhattacharya P, Neamtiu I (2011) Assessing programming language impact on development and maintenance: A study on c and c++, IEEE

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3(Jan):993–1022

Borges H, Brito R, Valente MT (2019) Beyond textual issues: Understanding the usage and impact of github reactions. In: Proceedings of the XXXIII Brazilian symposium on software engineering. pp 397–406

Cabot J, Izquierdo JLC, Cosentino V, Rolandi B (2015) Exploring the use of labels to categorize issues in open-source software projects, IEEE

Campbell JC, Zhang C, Xu Z, Hindle A, Miller J (2013) Deficient documentation detection a methodology to locate deficient project documentation using topic analysis, IEEE

Cardelli L, Wegner P (1985) On understanding types, data abstraction, and polymorphism. ACM Comput Surv (CSUR) 17(4):471–523

Chakraborty P, Shahriyar R, Iqbal A, Uddin G (2021) How do developers discuss and support new programming languages in technical q&a site? an empirical study of go, swift, and rust in stack overflow. Inf Softw Technol: 106603

Chen M, Fischer F, Meng N, Wang X, Grossklags J (2019) How reliable is the crowdsourced knowledge of security implementation?, IEEE

Chen TH, Thomas SW, Hassan AE (2016) A survey on the use of topic models when mining software repositories. Empir Softw Eng 21(5):1843–1919

Cifuentes C, Bierman G (2019) What is a secure programming language? In: 3rd Summit on advances in programming languages (SNAPL 2019), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

Cochran WG (2007) Sampling techniques. Wiley, Hoboken

Cohen J (1960) A coefficient of agreement for nominal scales. Educ Psychol Meas 20(1):37–46

Croft R, Newlands D, Chen Z, Babar MA (2021) An empirical study of rule-based and learning-based approaches for static application security testing. arXiv:2107.01921

Decan A, Mens T, Constantinou E (2018) On the impact of security vulnerabilities in the npm package dependency network. In: Proceedings of the 15th international conference on mining software repositories. pp 181–191

Dhurjati D, Kowshik S, Adve V, Lattner C (2003) Memory safety without runtime checks or garbage collection. In: Proceedings of the 2003 ACM SIGPLAN conference on language, compiler, and tool for embedded systems. pp 69–80

Fincher S, Tenenberg J (2005) Making sense of card sorting data. Expert Syst 22(3):89–93

Fischer F, Böttinger K, Xiao H, Stransky C, Acar Y, Backes M, Fahl S (2017) Stack overflow considered harmful? the impact of copy&paste on android application security. In: 2017 IEEE symposium on security and privacy (SP). IEEE, pp 121–136

Gkortzis A, Mitropoulos D, Spinellis D (2018) Vulinoss: a dataset of security vulnerabilities in open-source systems. In: Proceedings of the 15th International conference on mining software repositories. pp 18–21

Grossman D, Hicks M, Jim T, Morrisett G (2005) Cyclone: A type-safe dialect of c. C/C++ Users J 23(1):112–139

Han J, Shihab E, Wan Z, Deng S, Xia X (2020), What do programmers discuss about deep learning frameworks. Empir Softw Eng

Hanenberg S, Kleinschmager S, Robbes R, Tanter É, Stefik A (2014) An empirical study on the impact of static typing on software maintainability. Empir Softw Eng 19(5):1335–1382

Hata H, Treude C, Kula RG, Ishio T (2019) 9.6 million links in source code comments: Purpose, evolution, and decay, IEEE

Horschig S, Mattis T, Hirschfeld R (2018) Do java programmers write better python? studying off-language code quality on github. In: Conference companion of the 2nd international conference on art, science, and engineering of programming. pp 127–134

Howell DC (2012) Statistical methods for psychology. Cengage Learning

Hussain MM, Mahmud I (2019) pymannkendall: a python package for non parametric mann kendall family of trend tests, vol 4, p 1556. https://doi.org/10.21105/joss.01556

Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining github. In: Proceedings of the 11th working conference on mining software repositories (MSR 2014), pp 92–101

Khwaja AA, Murtaza M, Ahmed HF (2020) A security feature framework for programming languages to minimize application layer vulnerabilities. Secur Privacy 3(1):e95

Kleinschmager S, Robbes R, Stefik A, Hanenberg S, Tanter E (2012) Do static type systems improve the maintainability of software systems? an empirical study, IEEE

Knight WR (1966) A computer method for calculating kendall's tau with ungrouped data, vol 61, pp 436–439. http://www.jstor.org/stable/2282833

Kochhar PS, Wijedasa D, Lo D (2016) A large scale study of multiple programming languages and code quality. In: 2016 IEEE 23rd International conference on software analysis, evolution, and reengineering (SANER), IEEE, vol 1, pp 563–573

Le THM, Hin D, Croft R, Babar MA (2020) Puminer: Mining security posts from developer question and answer websites with pu learning. In: Proceedings of the 17th international conference on mining software repositories. Association for Computing Machinery, New York, pp 350–361. https://doi.org/10.1145/3379597.3387443

Le THM, Croft R, Hin D, Babar MA (2021) A large-scale study of security vulnerability support on developer q&a websites. In: Evaluation and assessment in software engineering. pp 109–118

Linares-Vásquez M, Dit B, Poshyvanyk D (2013) An exploratory analysis of mobile development issues using stack overflow, IEEE

Lopez T, Tun TT, Bandara A, Levine M, Nuseibeh B, Sharp H (2018) An investigation of security conversations in stack overflow: perceptions of security and community involvement. In: Proceedings of the 1st international workshop on security awareness from design to deployment. pp 26–32

Lopez T, Tun T, Bandara A, Mark L, Nuseibeh B, Sharp H (2019) An anatomy of security conversations in stack overflow. In: 2019 IEEE/ACM 41st international conference on software engineering: software engineering in society (ICSE-SEIS). IEEE, pp 31–40

Mailloux LO, Grimaila M (2018) Advancing cybersecurity: The growing need for a cyber-resiliency workforce. IT Professional 20(3):23–30

Mantyla MV, Claes M, Farooq U (2018) Measuring lda topic stability from clusters of replicated runs. In: Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement. pp 1–4

Martin A, Rashid A, Chivers H, Danezis G, Schneider S, Lupu E (2019) The cyber security body of knowledge. University of Bristol. https://www.cybok.org/

Massacci F, Nguyen VH (2010) Which is the right source for vulnerability studies? an empirical analysis on mozilla firefox. In: Proceedings of the 6th international workshop on security measurements and metrics. pp 1–8

Menezes AJ, Van Oorschot PC, Vanstone SA (2018) Handbook of applied cryptography. CRC press, Boca Raton

Meyerovich LA, Rabkin AS (2013) Empirical analysis of programming language adoption. SIGPLAN Not 48(10):1–18. https://doi.org/10.1145/2544173.2509515

Mimno D, Wallach HM, Talley E, Leenders M, McCallum A (2011) Optimizing semantic coherence in topic models. In: Proceedings of the conference on empirical methods in natural language processing. Association for Computational Linguistics. pp 262–272

Naiditch D (1999) Selecting a programming language for your project. IEEE Trans Aerosp Electron Syst 14(9):11–14

Nguyen VH, Massacci F (2013) The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. pp 493–498

Nothman J, Qin H, Yurchak R (2018) Stop word lists in free open-source software packages. In: Proceedings of workshop for NLP open source software (NLP-OSS). Association for Computational Linguistics, Melbourne, pp 7–12. https://doi.org/10.18653/v1/W18-2502. https://www.aclweb.org/anthology/W18-2502

Panichella A, Dit B, Oliveto R, Di Penta M, Poshynanyk D, De Lucia A (2013) How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms, IEEE

Pearson K (1900) X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. London, Edinburgh Dublin Philos Mag J Sci 50(302):157–175

Pierce BC, Benjamin C (2002) Types and programming languages. MIT press, Cambridge

Pletea D, Vasilescu B, Serebrenik A (2014) Security and emotion: sentiment analysis of security discussions on github. In: Proceedings of the 11th working conference on mining software repositories. ACM, pp 348–351

Rafter D (2019) 2019 data breaches: 4 billion records breached so far. [Online; accessed 01-June-2020]

Rahman MM, Roy CK (2014) An insight into the pull requests of github. In: Proceedings of the 11th working conference on mining software repositories. pp 364–367

Ray B, Posnett D, Filkov V, Devanbu P (2014) A large scale study of programming languages and code quality in github. In: Proceedings of the 22Nd ACM SIGSOFT international symposium on foundations of software engineering (FSE 2014). ACM, New York, pp 155–165. https://doi.org/10.1145/2635868.2635922

Rosen C, Shihab E (2016) What are mobile developers asking about? a large scale study using stack overflow. Empir Softw Eng 21(3):1192–1223

Seacord RC (2005) Secure coding in C and C++. Pearson Education, London

Sebesta RW (2012) Concepts of programming languages. Pearson Education, Inc., London

Sestoft P (2005) Java performance: Reducing time and space consumption

Shahriar H, Zulkernine M (2012) Mitigating program security vulnerabilities: Approaches and challenges. ACM Comput Surv (CSUR) 44(3):1–46

Sindre G, Opdahl AL (2005) Eliciting security requirements with misuse cases. Req Eng 10(1):34–44

Tian Y, Ng W, Cao J, McIntosh S (2019) Geek talents: Who are the top experts on github and stack overflow? Comput Mater Contin 61(2):465–479. https://doi.org/10.32604/cmc.2019.07818. http://www.techscience.com/cmc/v61n2/33484

Treude C, Wagner M (2019) Predicting good configurations for github and stack overflow topic models. In: 2019 IEEE/ACM 16th International conference on mining software repositories (MSR). pp 84–95

Vasilescu B, Filkov V, Serebrenik A (2013) Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: 2013 International conference on social computing. pp 188–195. https://doi.org/10.1109/SocialCom.2013.35

Venson E, Alfayez R, MF GM, Rejane F, Boehm B (2019) The impact of software security practices on development effort: An initial survey. In: 2019 ACM/IEEE international symposium on empirical software engineering and measurement (ESEM). IEEE, pp 1–12

Verdi M, Sami A, Akhondali J, Khomh F, Uddin G, Motlagh AK (2020), An empirical study of c++ vulnerabilities in crowd-sourced code examples. IEEE Trans Softw Eng

Wan Z, Xia X, Hassan AE (2019) What is discussed about blockchain? a case study on the use of balanced lda and the reference architecture of a domain to capture online discussions about blockchain platforms across the stack exchange communities. IEEE Trans Softw Eng

Wessel M, Steinmacher I, Wiese I, Gerosa MA (2019) Should i stale or should i close? an analysis of a bot that closes abandoned issues and pull requests, IEEE

Xiong Y, Meng Z, Shen B, Yin W (2017) Mining developer behavior across github and stackoverflow. In: SEKE. pp 578–583

Yang L, Li X, Yu Y (2017) Vuldigger: A just-in-time and cost-aware tool for digging vulnerability-contributing changes. In: GLOBECOM 2017-2017 IEEE global communications conference. IEEE, pp 1–7

Yang XL, Lo D, Xia X, Wan ZY, Sun JL (2016) What security questions do developers ask? a large-scale study of stack overflow posts. J Comput Sci Technol 31(5):910–924. https://doi.org/10.1007/s11390-016-1672-0

Zahedi M, Ali BabarM, Treude C (2018) An empirical study of security issues posted in open source projects. In: Proceedings of the 51st Hawaii international conference on system sciences

Zhang H, Wang S, Li H, Chen THP, Hassan AE (2021) A study of c/c++ code weaknesses on stack overflow. IEEE Trans Softw Eng

Zhang J, Li F, Hao D, Wang M, Tang H, Zhang L, Harman M (2019) A study of bug resolution characteristics in popular programming languages. IEEE Trans Softw Eng

Zou J, Xu L, Guo W, Yan M, Yang D, Zhang X (2015) Which non-functional requirements do developers focus on? an empirical study on stack overflow using topic analysis. In: 2015 IEEE/ACM 12th working conference on mining software repositories, IEEE. pp 446–449

Affiliations

Roland Croft[1,2] ⓘ **· Yongzheng Xie**[1] **· Mansooreh Zahedi**[3] **· M. Ali Babar**[1,2] **· Christoph Treude**[3]

Yongzheng Xie
Yongzheng.Xie@adelaide.edu.au

Mansooreh Zahedi
mansooreh.zahedi@unimelb.edu.au

M. Ali Babar
Ali.Babar@adelaide.edu.au

Christoph Treude
christoph.treude@unimelb.edu.au

[1] School of Computer Science, The University of Adelaide, Adelaide, Australia

[2] Cyber Security Cooperative Research Centre, Perth, Australia

[3] School of Computing & Information Systems, The University of Melbourne, Melbourne, Australia