The Effect of Google Search on Software Security: Unobtrusive Security Interventions via Content Re-ranking

Felix Fischer, Yannick Stachelscheid, Jens Grossklags
Technical University of Munich
{flx.fischer, yannick.stachelscheid, jens.grossklags}@tum.de

ABSTRACT

Google Search is where most developers start their Web journey looking for code examples to reuse. It is highly likely that code that is linked to the top results will be among those candidates that find their way into production software. However, as a large amount of secure and insecure code has been identified on the Web, the question arises how the providing webpages are ranked by Google and whether the ranking has an effect on software security.

We investigate how secure and insecure cryptographic code examples from Stack Overflow are ranked by Google Search. Our results show that insecure code ends up in the top results and is clicked on more often. There is at least a 22.8% chance that one out of the top three Google Search results leads to insecure code.

We introduce security-based re-ranking, where the rank of Google Search is updated based on the security and relevance of the provided source code in the results. We tested our re-ranking approach and compared it to Google's original ranking in an online developer study. Participants that used our modified search engine to look for help online submitted more secure and functional results, with statistical significance. In contrast to prior work on helping developers to write secure code, security-based re-ranking completely eradicates the requirement for any action performed by developers. Our intervention remains completely invisible, and therefore the probability of adoption is greatly increased. We believe security-based re-ranking allows Internet-wide improvement of code security and prevents the far-reaching spread of insecure code found on the Web.

CCS CONCEPTS

• **Security and privacy** → **Usability in security and privacy**; • **Information systems** → *Content ranking*.

KEYWORDS

usable security; software development; Web search; content ranking

ACM Reference Format:
Felix Fischer, Yannick Stachelscheid, Jens Grossklags, *Technical University of Munich*, {flx.fischer, yannick.stachelscheid, jens.grossklags}@tum.de . 2021. The Effect of Google Search on Software Security: Unobtrusive Security Interventions via Content Re-ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3484763

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

1 INTRODUCTION

"Just Google it!" Using the dominant search engine has become one of the most popular ways to find an answer to a question when there is no easy explanation at hand. This counts for software developers as well, as it is often the first stop on their journey to solve urgent programming problems. This is because the Web offers everything developers need to know; from books and tutorials to documentation and code examples. As such, solutions seem to be only a few queries and clicks away.

Even though Google Search provides thousands of results for a single query, users typically choose a link that is among the top results, or "above the fold."[1] If there is nothing relevant, they try another query and Google again. That means search results that are above the fold may have a tremendous impact on today's software as they define what developers learn and how they solve problems.

Researchers at Google have shown that developers mostly used their search engine in order to find out how to use a specific application programming interface (API) and to find functional examples [39]. However, the safe use of cryptographic APIs not only heavily relies on functional examples but on the availability of secure best-practice examples [1, 2, 17, 18, 22]. Since the use cases of these APIs are very different and complex [17, 18], Google Search appears to be essential for finding the right examples for the problem at hand [2].

Stack Overflow is among developers' most favorite Web resource [2] and provides a huge amount of ready-to-use code examples [18]. Prior work shows that while developers usually start with Google Search [2, 39, 41], Stack Overflow is almost always part of the subsequent Web journey [2].

Unfortunately, these code examples often provide very weak cryptography or contain severe software vulnerabilities. Moreover, software developers tend to reuse these code examples over secure ones even though sometimes significantly more secure than insecure examples are available. Ninety-seven percent of Google Play applications that reused code from Stack Overflow reused at least one insecure example [17].

This phenomenon might be explained by a generally higher Google Search rank for insecure code. The Google Search rank is a very powerful positive indicator that people tend to follow even if the displayed abstract of the related result is less relevant [32]. This selection bias might contribute to the reuse of insecure code if it ends up in the top results.

If this is the case, could this observation be harnessed to improve software security by a deliberate and comprehensive re-ranking

[1] https://backlinko.com/google-ctr-stats

of search results? To investigate this important question, we have perform an end-to-end investigation on whether modifications to the rank have a direct significant effect on the security of code written by software developers.

First, in an online study, we have systematically tested how secure and insecure cryptographic code examples are currently ranked by Google Search. We collected search results from 192 developers and analyzed the security of Stack Overflow code examples found among the top ten search results (t_{10}) that are shown on the first page of Google Search.

We find that significantly more results in t_{10} provide links to insecure code. More interestingly, if t_{10} provides an insecure link chances are higher that it is among the top three results. In fact, there is a 22.8% probability that *one out of the three* top results link to insecure code examples. Further, there is only a 46.1% chance that users will not encounter an insecure Stack Overflow result on Google Search.

Second, we introduce and evaluate *security-based re-ranking*. It applies a semi-supervised clustering method to identify code examples on the Web that are not only secure, but provide secure *best practices*. This means they provide the secure end-to-end pattern to solve the given use case at hand. By boosting these examples in search results and decreasing the rank of insecure results, we have improved the security distribution of results in t_{10}. Afterwards, we observed a *near zero probability* that the top three results contained insecure code. Moreover, there was a 29.4% chance that at least one top three result contained secure best practices.

Lastly, we tested security-based re-ranking in an online developer study with 218 participants that had to solve several security-related programming tasks. The more the participants interacted with the modified search engine, the more secure and functional solutions they submitted. We show that this effect is statistically significant ($p < 0.05$). We did not observe this effect from our control group using original Google Search, who provided more insecure solutions.

We summarize our contributions as follows:

- We demonstrate that Web search ranking has a significant effect on software security.
- We show that the current distribution of insecure coding practices in the top results of Google Search is significantly higher than those of secure ones.
- We have developed security-based re-ranking which helps in identifying secure best practices on the Web and adjusts the Google Search ranking to show these results preferentially.
- We have performed a developer study and show that security-based re-ranking significantly helps software developers to write more secure code in comparison to those that used original Google Search.

We structure and present our work as follows. We first discuss related work in the domains of search and security, software developer studies, and developer search behaviors in Section 2. We then describe our first online study in Section 4, where we measure the security of search results and present the results in Section 4.5. In Section 5, we continue with the presentation of our methodology for security-based re-ranking, and follow with its evaluation in Section 5.6. Finally, we test whether security-based re-ranking has an effect on code security in our second online study in Section 6 and present the results in Section 6.2.

We support open science and open-source our data including surveys, study tasks, source code, and results.[2]

2 RELATED WORK

2.1 Search and Security

A key area of prior work relates to understanding search behavior. Mining query data from search engines suggests that patterns follow a power-law distribution [5], i.e., there are huge differences in the importance of the quality of results. Likewise, research over the last 20 years has shown that individual consumers typically visit only a limited set of — often prominent — distinct websites [10, 12].

To protect users from reaching unwanted search destinations, various approaches have been taken, ranging from automatic detection of "dangerous" websites (e.g., [30, 45]) to using notifications to inform users about such risks, e.g., Google Safe Browsing [21].

From a privacy perspective, earlier work focused on embedding P3P-related indicators in the search engine [8]. In the security context, a variety of indicators have been studied regarding their effectiveness, in particular in the phishing context [13, 14, 16, 26].

While the extant literature for this topic space is tremendously rich, we are unaware of research on the impact of common Internet search engines on the utilization of formal or informal security advice sources, and the eventual implications for code security.

2.2 Software Developer Studies

In the past five years, the research area of usable security has increasingly focused on the work practices of developers [3], thereby complementing the existing literature on end-user security practices and behaviors [38, 50].

Previous work has shown that developers do not only vary in their ability to deliver functional code, they also vary in terms of creating code free of security problems [4].

In this context, programming advice forums play an important role when it comes to code delivered by developers. For example, previous work has shown that 30% of cryptographic code examples on Stack Overflow were insecure, and that these insecure samples were reused in over 190,000 Android apps [17]. Bai et al. performed a survey to find out why developers reuse insecure code [6]. A user study has shown that programmers prefer programming advice platforms over more traditional sources such as textbooks and official programming documentation [2].

In fact, a developer survey (with over 88,000 software developers from 179 countries) conducted by Stack Overflow reported that over 60% visit the platform at least once or more every day, and that 96% come to seek solutions to specific problems [42].

Identifying secure code on Stack Overflow is challenging, since quality signals can often be misleading. Based on the manual inspection of a large sample of advice postings for cryptographic code, it was shown that *insecure* posts were most often associated with higher view counts and scores, and were frequently posted

[2] https://github.com/TUM-ChairOfCyberTrust/security-based-reranking.git

by so-called trusted users [9]. Another study showed that additional information beyond the code can be taken into consideration to identify insecure code on Stack Overflow with machine learning methods [55]. Other research documented that developers are easily influenced by the high-level (irrelevant) appearance of postings, rather than such quantitative ratings provided on Stack Overflow [44]. This means even developers may not even always follow quantitative indicators.

Several studies have investigated how to help developers to deliver more secure code. On the one hand, multiple systems such as FixDroid have been designed to support developers with easy-to-use tools to check (cryptographic) code and to provide fixes [31].

On the other hand, several works address the complexity of developing secure code. One solution approach is to provide developers with simplified APIs to prevent incorrect usage [1]. However, the associated developer study showed that code sometimes lacked necessary functionality, and the simplified APIs were not applicable to several specific use cases.

Recent work has tried to harvest the existing wealth of information on Stack Overflow more directly by using machine learning techniques to distinguish secure from insecure postings on cryptographic use cases, and reworking the Stack Overflow interface to nudge users to postings which are relevant for the use case and are secure [18]. A user study shows that developers produce functional code, which is often more secure using the improved system.

Our review of publications in the security and usable security space did not yield any research that specifically addresses how developers search for secure solutions with search engines.

2.3 Developer Search Behavior

However, a variety of studies have been conducted to explore developer search behaviors in the software engineering domain. The studies analyze how developers use specialized code search engines and also Web search engines.

A study with developers at a large Internet company showed that they used a specialized code search engine for the internal codebase 12 times each day on average [39]. The study used an experience sampling methodology. Another recent study explored how developers use Web search engines with a focus on comparing code search with other search behaviors [35]. The authors explored, e.g., whether the used vocabulary or the length of queries differ across the two contexts. In another survey study, it was found that 91% (out of 55 developers) had used search engines in the past to look for source code examples. However, directly using search functionality on "social help sites" (e.g., StackOverflow, Quora etc.) had only been done by 36% of the participants [27], which emphasizes the importance of the Web search context in our work.

A multi-method study investigating survey responses of 235 software engineers as well as the search queries of 60 developers sheds some light on the self-reported frequency and perceived difficulty of searching for certain types of programming-related information on the Web, as well as actual query frequencies [53]. Somewhat relevant to our work, participants reported searching for "explanations for exceptions/error messages (e.g., HTTP 404)" and "solutions to common programming bugs" *often*, and considered these search tasks to be relatively *easy*. The participants further reported to *sometimes* "search for reusable code snippets" (with a *neutral* level of difficulty). The analysis of the query logs suggests that these three categories belong to the most frequent search activities. The study, however, remains at a very high level, and in particular does not show whether developers are able to identify secure search results, and whether their searches contribute to the development of functional and secure code.

Our work builds on the existing literature of developer studies. However, to the best of our knowledge, no study exists that focuses on security search behavior, or investigates the impact of using search engines on code security.

3 ONLINE STUDIES

We performed two different online studies: *Security of Search Results* and *Impact of Ranking on Code Security*.

In Study 1 on *Security of Search Results*, 192 participants performed code search on the Web using Google Search. We calculated the distribution of Stack Overflow results that provide secure code examples and the distribution of Stack Overflow results that provide insecure ones among the top ten results t_{10}. Distributions were calculated for a set of 274 distinct user queries Q and 3,800 related search results R.

Further, we wanted to know whether the ranking of search results actually has an impact on code security. In Study 2, 218 participants had to solve three security-related programming tasks on the *Impact of Ranking on Code Security* with the help of Google Search. Therefore, we tested two conditions. The control group had to use the original Google Search engine that was used in Study 1, while our treatment group used a different Google Search engine that applies *security-based re-ranking*. We developed *security-based re-ranking* in order to up-rank secure results and down-rank insecure results. We explain this approach in Section 5.

Age				
Mean = 32.03/30.94	Median = 29.5/29	Stddev = 10.34/8.59	Min = 18	Max = 74/59
Country of Origin				
USA = 40/60	Germany = 23/19	Brazil/India = 13/18	China/Brazil = 11/9	Other = 105/112
Gender				
Male = 171/188	Prefer not to say = 5/22	Other = 6/1		Female = 10/7
Level of Education Achieved				
High School = 30/21	Bachelor= 80/102	Master = 44/60	PhD = 18/11	Other = 20/24
Professional				
Yes = 133/161	No = 52/33			N/A = 7/24
Security Background				
Yes = 56/46	No = 129/143			N/A = 7/29
Java Years of Experience				
< 1 year = 51/61	1 to 2 years = 35/40	> 2 years = 82/80		N/A = 25/37
Java Primary Focus of Job				
Yes = 31/49	No = 158/144			N/A = 3/25

Table 1: Detailed data about demographics of participants for Study 1 (N = 192) and Study 2 (N = 218).

3.1 Recruitment

We recruited participants by contacting GitHub developers via email. We extracted around 900,000 email addresses from public GitHub user profiles. A randomly compiled list of 50,000 users was contacted for Study 1. To incentivize users, ten Amazon vouchers worth 50 USD were given away in a drawing for all participants. A randomly selected list of 100,000 users was contacted for Study 2. Here we offered no compensation. We attached an opt-out link to

all of our study emails that (if used) deleted the recipient's email address from our database.

Participants were first directed to a landing page that introduced the respective study, followed by a comprehensive consent form. We avoided priming in the description of the study and tasks by not mentioning any terms related to security or privacy. After completing the main task, participants were asked to complete a short exit survey that explored the characteristics of the specific sample. In Study 1, 192 out of 991 participants that entered the landing page completed the survey. In Study 2, out of the 827 participants that began the survey, 218 finished it.

3.2 Research Ethics

Our institution does not require IRB review for online survey studies. However, we closely followed the recruitment and remuneration modalities of a GitHub developer study (discussed in the related work section), which had received approval by the Ethics Review Board of Saarland University, the IRB of the University of Maryland, and the NIST Human Subjects Protection Office [4].

Survey studies with GitHub users as the survey population are relatively common. Most prominently, the 2017 GitHub Open-Source Survey provided detailed knowledge about the user population on the platform [19, 20]. In the security domain, we are aware of one study using GitHub as the primary participant source [4, 22]. In contrast, there are numerous peer-reviewed studies in the domain of software engineering that survey GitHub developers to explore a variety of coding and development practices [7, 23, 24, 28, 34, 36, 37, 40, 46, 52] as well as gender and diversity issues [47, 48]. Most of those papers used email for recruitment.

3.3 Demographics, Experience, and Education

We collected demographic data about our study participants using an exit survey and present them in Table 1. Participants were asked about general information including country of origin, age, and gender. In order to control our samples for programming expertise, we asked for developer-specific information such as years of experience in Java and whether they were professionals. Lastly, we collected security-related information such as security experience and background.

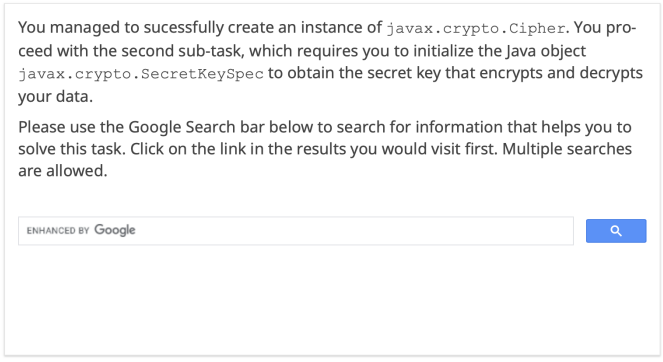


Figure 1: Description of a sub-task in Study 1

4 STUDY 1: SECURITY OF SEARCH RESULTS

4.1 Tasks

In order to measure the distribution of insecure Stack Overflow links in the top ten results t_{10}, we performed an online study where participants had to perform several queries using Google Search. They were introduced into a scenario setting where they had to solve hypothetical programming tasks on the topic of AES encryption. We followed our prior work (Fischer et al. [18]) in the design of the tasks, which include initializing a symmetric cipher (CIPHER), an initialization vector (IV), and a cryptographic key (KEY). Each task had several sub-tasks, e.g., inform yourself about symmetric encryption or look up how to use a specific API element. We show a screenshot of a sub-task in Figure 1.

Participants were instructed to type a search query into the provided search bar for each task in order to find help online. Next, they had to select the result that appeared to be the most relevant to solve the task. To decide relevance, participants could visit the webpage of each result and make a final decision. Thereby, we stored the search queries, the links from the results including their rank, and a click log.

4.2 Setup

The study was performed on the participants' own devices. The search bar was run by a Google Custom Search Engine (CSE) that we integrated into the online survey. The CSE API allows for different types of search customization, including filtering and boosting results. This way, we were able to store all inputs and outputs and to control what was shown in the results.[3] Results were not affected by Google's search personalization [25], since those services are not supported by CSE.[4]

4.3 Ground Truth

To be able to determine the security of search results, we needed ground truth about the security of code examples that are provided by Stack Overflow webpages shown in the results. We used TUM-Crypto,[5] an open-source dataset which contains security labels for the complete list of Java code examples on Stack Overflow that provide potential solutions for our study tasks CIPHER, IV, and KEY [18]. Each code example from the dataset provides a link to the Stack Overflow webpage it was downloaded from. This allowed us to label the security of Stack Overflow links that are relevant for solving the tasks.

4.4 Security Labeling

We defined the webpage labeling function l_{t1}, which labeled a Stack Overflow webpage as insecure if the top answer given on the webpage contained insecure code. A webpage was labeled as secure if otherwise. Measuring the distribution for l_{t1} gave us a realistic view on the distribution of results, that are more likely of being reused in production code. Chen et al. [9] have shown that code examples from the top answer are significantly more often reused than examples with a lower score. That is probably due to

[3] We removed all ads that would have been shown to the user.
[4] https://support.google.com/programmable-search/answer/70392?hl=en
[5] https://github.com/fischerfel/TUM-Crypto

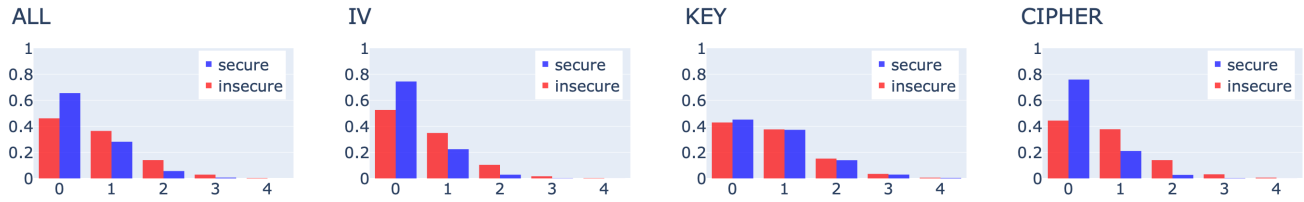


Figure 2: Binomial distribution of secure (blue) and insecure (red) Stack Overflow results over the top ten results t_{10}. The x-axis represents the observed count of secure or insecure results in t_{10}. The y-axis shows the probability for each observation, i. e. there is a probability of y that x results are secure or insecure in t_{10}.

the top answer the first one being seen by the user as it is shown right below the question. Further, it oftentimes comes with a green check mark that highlights the accepted answer by the user who posted the question.

4.5 Results

The TUM-Crypto dataset contains 3,361 secure and 7,195 insecure code examples, which are provided in 5,811 Stack Overflow discussion threads (i.e., webpages). For webpages that provide code examples for CIPHER, IV, and KEY, we obtained 1,671 secure and 2,057 insecure ones after applying our webpage labeling function l_{t1}, described in Section 4.4. These webpages represent a subset of the potential search results R.

Participants that typed a search query into our CSE obtained search results from $R = \{cse(q)\}_{q \in Q}$, where Q is the complete set of queries we tracked during the study and *cse* the search function implemented in our CSE. Note that R provides all search results returned by the search engine and was not subject to any modifications. However, only those results that could be linked to TUM-Crypto obtained a security label.

On average, our 192 participants submitted 1.3 queries and performed 1.0 clicks per task.

We obtained $|Q| = 274$ unique queries that were typed into the CSE. We observed 98 unique queries for IV, 87 for KEY, and 89 for CIPHER resulting into 1,290 top ten search results for IV, 1,210 for KEY, and 1,300 for CIPHER.

Stack Overflow was the most clicked on domain with 32% of clicks followed by official documentation websites for Java (docs.oracle.com) with 26%, and Android (developer.android.com) with 10% of the clicks. Rank one and two were the most clicked on ranks with both 33% of the clicks and the top three results received 79% of the clicks made in total.

Binomial Distribution— Based on the collected queries Q and results R from our three tasks, we measured the probability of secure and insecure Stack Overflow results to appear in the top three (t_3), top five (t_5) and top ten (t_{10}) results.

We calculated the binomial distribution $B(n, p)$ of secure and insecure results over $t_n \in R$, where $n \in \{3, 5, 10\}$ is the number of trials, i. e., the number of displayed results in t_n, and p the success probability of each trial, i. e., probability of a result to be secure or insecure. The probability is given by $p = r/s$, where r is either the number of secure or insecure Stack Overflow results in t_n, and r the total number of results in t_n. The distributions are calculated simulating 10,000 searches.

Second, we calculated the task-specific distributions of secure and insecure results over $R_{iv} = \{cse(q)\}_{q \in Q_{iv}} \subset t_n$ where Q_{iv} is the set of queries entered into the search bar during the IV task. Likewise, we measured the distributions of secure and insecure results over $R_{key} = \{cse(q)\}_{q \in Q_{key}} \subset t_n$ and $R_{cip} = \{cse(q)\}_{q \in Q_{cip}} \subset t_n$ where Q_{key}, Q_{cip} are the set of queries entered during the KEY and CIPHER task.

Figure 2 shows the binomial distribution over aggregated results from all three tasks, and over results from each individual task.

Task-Independent Results— Figure 2, *(ALL, t_{10}, blue)*, shows the distribution $B(10, 0.030)$ of secure results, while *(ALL, t_{10}, red)* shows the distribution $B(10, 0.070)$ of insecure results over t_{10} for the complete set of queries Q. We observed a 28.17% chance for at least one secure result to appear in t_{10} vs. a chance of 36.47% for one insecure result. Probabilities of more than one result to appear in t_{10} were also much higher for insecure results: 14.07% for two, 2.91% for three, and 0.32% for four vs. 5.22% for two and 0.64% for three secure results to be found in t_{10}.

Probabilities diverged even stronger in the top three results t_3. Here, we observed a chance of 22.78% for at least one insecure result where a secure result has only a 9.19% chance. This means if an insecure result ends up in t_{10} it is more likely that it obtains a rank in the top three than in the lower ranks. Further, every fourth Google Search query will probably show an insecure result in t_3 if the query is related to IV, KEY, or CIPHER.

IV— Figure 2, *(IV, t_{10}, blue)* contrasts the distribution $B(10, 0.004)$ of secure results with the distribution $B(10, 0.060)$ of insecure results shown in Figure 2, *(IV, t_{10}, red)* for queries $q \in Q_{iv}$.

We observed a 22.44% chance that at least one result in t_{10} is secure vs. a 34.90% chance for one insecure result. While probabilities were near 0% for a secure result to be contained in t_5 or t_3, we observed 29.79% for t_5 and 25.10% for t_3 to provide at least one insecure result. In contrast to secure results, insecure results also had a chance for more than one result to be present in t_{10}: 10.43% for two, and a 1.65% for three results.

KEY— We observed a surprisingly high distribution of secure results in t_{10} for queries $q \in Q_{key}$. There was a 37.32% chance for at least one secure result (see Figure 2, *(KEY, t_{10}, blue)*). The probability of an insecure result was only slightly higher with 37.65%, as shown in Figure 2, *(KEY, t_{10}, red)*.

Probabilities of more than one result in t_{10} were also higher for insecure results: 15.22% for two, and 3.47% for three vs. 13.99% for two, and 2.94% for three secure results.

Once more, probabilities of one insecure result in t_5 and t_3 only slightly decreased with 37.03% and 29.19%, respectively.

Even though we observed a lower probability of secure results in t_{10} again, the probability values do not differ as much as observed in IV. However, probabilities diverged more significantly in t_3 again, where one insecure result has a chance of 29.19% and a secure result 19.48%.

CIPHER— For CIPHER, there was a 21.06% chance for at least one secure result in t_{10} and a 2.71% chance for two results (see Figure 2, *(CIPHER, t_{10}, blue)*). We only observed a 3.26% chance for one result in t_3 to be secure.

The distribution of insecure results shown in Figure 2, *(CIPHER, t_{10}, red)* is not very different from IV and KEY. $B(10, 0.077)$ revealed a 37.77% chance for at least one result, 14.05% for two, and 3.16% for three results. We observed very similar probabilities of insecure results in t_5 and t_3 as observed for IV and KEY. $B(5, 0.089)$ revealed a 30.24% chance and $B(3, 0.065)$ a 17.70% chance for one insecure result to be in t_5 and t_3, respectively.

Summary— Study 1 has shown that the distribution of insecure Stack Overflow results in the top tiers t_3, t_5, and t_{10} is significantly higher than the distribution of secure Stack Overflow results. This was the case for all three tasks that were performed by our participants. Moreover, the probability of insecure results to end up in t_3 is higher than for lower ranks.

These observations led to the following research questions: Does the higher ranking of insecure Stack Overflow code examples actually have a negative impact on code security? Further, does security-based re-ranking help in mitigating this effect by down-ranking insecure and up-ranking secure results? We first present our approach for security-based re-ranking in the following Section 5 and investigate both research questions in Study 2 in Section 6.

5 SECURITY-BASED RE-RANKING

In Section 4.5, we have shown that Google Search currently up-ranks Stack Overflow results that provide insecure cryptographic code. We introduce security-based re-ranking to tackle this problem. In a nutshell, based on the security label of a webpage's content (i. e., source code), its rank is updated by either lowering or raising it.

This would help developers to start their Web search on a safe path. In fact, the Web offers secure best-practice examples that help developers to finish their programming tasks functionally and securely for a wide range of use cases [18]. Boosting those examples in Google Search while simultaneously lowering the rank of insecure results could have a ripple effect on the code security of production software.

In this section, we describe how we identified secure best-practice examples in TUM-Crypto and how we re-ranked results based on security. We continue to use the labeling function l_{t1} to determine the security of webpages.

5.1 Secure Best-Practice Examples

A *secure* Stack Overflow webpage is labeled as a best-practice result if it is secure with respect to l_{t1} *and* provides at least one best-practice example for IV or KEY. An example is a code snippet which gives the complete API usage pattern for how to safely generate a cryptographic key or an IV (see Figure 9 in the Appendix). We distinguish *secure best-practice* from *secure* examples. A secure code example is *not* a best-practice example if the API usage pattern is incomplete. This means that the example does not show how to safely implement all necessary dependencies of IV or KEY. For instance, it does not show how to initialize a secure random number generator which is necessary to generate a cryptographic key. In practice, popular (but misleading) examples are code snippets that show how to encrypt a string using a key that is passed through a method parameter. Even though these examples may be secure (in terms of not being insecure), developers will not learn the complete pattern since key generation is missing. We do not make this differentiation for CIPHER since initializing a CIPHER using the Java SDK is done within a single statement that does not have any dependencies.

We will show in later sections that patterns are incomplete for most secure KEY and IV examples on Stack Overflow. This causes the following problem: developers may be presented with secure but unhelpful code. This forces them to continue the code search, potentially leaving the secure path, ending up reusing insecure code once again. By up-ranking secure best practices, there is higher probability that developers start and stay on a safe path.

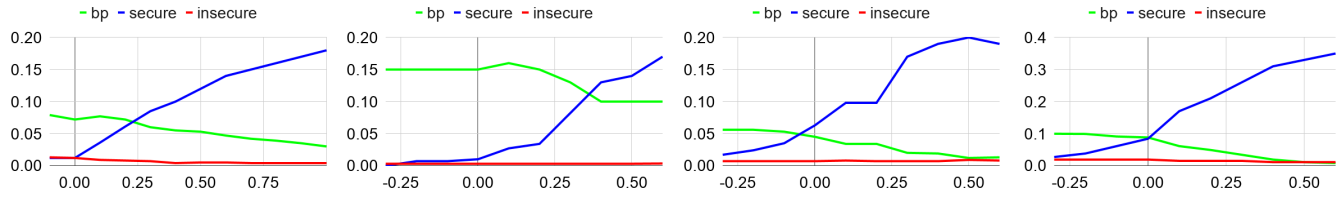
5.2 Code Embeddings

We have followed a semi-supervised methodology to identify secure best-practice examples in TUM-Crypto. We first applied DBSCAN, an unsupervised clustering method to cluster secure API usage patterns for the use cases KEY and IV. Those provide the most diverse patterns available in the dataset as shown in [18]. Note, our clustering method does not separate secure from insecure patterns, but is solely applied to secure ones to separate best practices from potentially less helpful examples.

The advantage of DBSCAN is that it does not need to know the number of expected clusters in advance. Since its clustering method is performed on input vectors, we need vector representations for the API usage patterns. In Fischer et al. [18], we have trained a deep learning model that generated embeddings for API usage patterns from TUM-Crypto. These embeddings were learned such that similar patterns are close, and dissimilar ones are far away from each other in the embedding space, using cosine distance as the distance metric. The learned embeddings encode data and control dependencies of code graphs, as well as lexical information of code statements. We used them as inputs for DBSCAN because it supports clustering based on cosine distance. Please refer to [18, 54] for further details on the generation of the embeddings.

5.3 Best-Practice Clustering

DBSCAN automatically creates clusters based on two parameters: the maximum distance ϵ of an embedding to the core sample of the cluster, and the minimum number of embeddings $minPts$ necessary to form a cluster. It does not require the number of clusters to be



(a) IV, t_{10} **(b) IV, t_3** **(c) KEY, t_{10}** **(d) KEY, t_3**

Figure 3: Parameter search for boosting results for IV and KEY. The horizontal axis represents boost values and the vertical axis the probability of a secure best-practice or a secure or insecure result to appear in the top ten results.

known in advance as it is determined based on these parameters. Embeddings that cannot be assigned to a cluster end up in the noise cluster N.

After we obtained the set of clusters $C := DBSCAN(\epsilon, minPts) \setminus N$ for a given pair of ϵ and $minPts$, we looked at exactly on representative c_i for each cluster $C_i \in C$ to determine whether it is a secure best practice, or not. This way manual labeling effort is given by cardinality $|C|$, and its reduction r_{lab} given by $r_{lab} = 1 - |C|/|E|$, where E is the set of embeddings generated from all samples in TUM-Crypto.

We wanted to keep the manual labeling effort as low as possible and therefore r_{lab} close to 1. Additionally, we wanted the clustering to be as accurate as possible in order not to miss any best-practice examples. Therefore, the size of the noise cluster $|N|$ needs to be as small as possible, as well as the biggest cluster $C_{max} \in C$. These three objectives are competing with each other. Smaller $|N|$ leads to more clusters $|C|$ and to smaller r_{lab}.

It follows that we had to solve the following multi-objective optimization task: Choose parameters ϵ and $minPts$ that maximize r_{lab}, while minimizing the percentage of the biggest cluster $|C_{max}|/|E|, |C_{max}| \geq |C_i|, \forall C_i \in C$ and the percentage of the noise cluster $|N|/|E|$.

The smaller $|C|$, the smaller the manual effort r_{lab}. The bigger C_{max}, the higher the chance ϵ has been chosen too large; thereby insufficiently differentiating between API patterns. If $|N|$ becomes too large, it might contain best-practice examples that would be missed since the noise cluster naturally will not be considered for manual inspection.

5.4 Pareto Optimum

We determined the optimal parameter by performing a grid search for a Pareto-optimal solution. Thereby, we calculated all values for labeling effort, noise, and maximum cluster size during the grid search with the objective to minimize them. The grid was given by $\epsilon \in [0.0, 0.09]$ as we observed that $|C|$ quickly converged to 1 for bigger cosine distances than 0.09. We searched for $minPts \in [2, 10]$, as we only needed 10 best-practice examples to show in the top ten tier of the Google Search results. Our search space was very small. ϵ was increased by 0.01 for each search, resulting in a grid with dimension 10×9.

We calculated the knee-points for the $3d$ Pareto curve for all three competing objectives and derived the related DBSCAN parameters for IV and KEY. The results are shown in Table 3 in the Appendix. Distance ϵ was similar for both use cases with 0.3 and 0.4 and both had a minimum cluster size of two. For both use cases, we reached a noise level of around 10% of the respective example set. Those examples were not considered for manual labeling.

We randomly selected a representative for each cluster and manually reviewed the source code of the representative to decide whether it is a best-practice example. If yes, all samples from the related cluster were added to the set of best-practice examples of the related class IV or KEY, respectively. The manual labeling effort was below 10%, which resulted in 65 samples for IV and 114 samples for KEY.

Finally, we evaluated the precision of the whole process. We selected a maximum of 50 random samples from each secure best-practice cluster and manually evaluated whether they were true positives. We obtained a precision of 0.81 for KEY and 0.98 for IV. None of the false positives were insecure.

5.5 Re-ranking

We implemented a CSE that updates the rank of Stack Overflow results based on the security of the content and whether it provides best-practice examples. The CSE API provides functionality to influence the ranking of search results. Those modifications are domain-based. For a given URL, attributes can be defined in order to boost their rank. For a given Stack Overflow link, our CSE assigns the search function cse_b a boost $b \in [-1, 1]$.

Parameter Search— Trivially, we set the boost value b_{bp} for secure best-practice results to the maximum of $b_{bp} = 1.0$ and for insecure results to the minimum of $b_i = -1.0$. We performed a parameter search to find the optimal boost value b_s for secure results that were not best practices. On the one hand, one would expect those results to obtain a lower rank than secure best practices, since they may be less helpful. On the other hand, one would require them to have a higher rank than insecure results.

Therefore, they naturally compete with secure best-practice and insecure results to obtain ranks in t_{10}. If the boost for secure results is too high, some secure best-practice results in t_{10} may be replaced by secure results. If the boost is too low, some secure results may be replaced by insecure ones. Therefore, the boost for secure results has to be high enough to push insecure results further out of t_{10} while not interfering with secure best-practice results.

To find the optimal boost value for secure results, we repeated our measurements from Study 1 (see Section 4). We calculated probabilities $p = r/s$ for secure best-practice (p_{bp}), secure (p_s), and

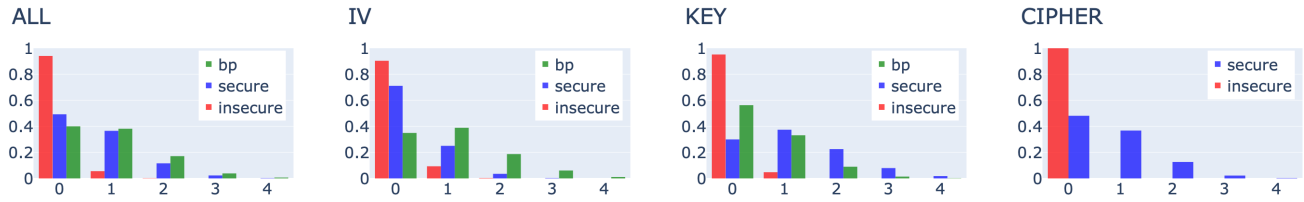


Figure 4: Binomial distribution of boosted secure best-practice (green), secure (blue), and insecure results (red) for IV, KEY, and CIPHER over t_{10}. The x-axis represents the observed count of secure best-practice, secure, or insecure results in t_{10}. The y-axis shows the probability for each observation, i. e., there is a probability of y that x results are secure best-practice, secure, or insecure in t_{10}.

insecure results (p_i) to appear in t_{10}. Here, r was either the number of secure best-practice, secure, or insecure results in t_{10}, and s the total number of results in the respective tier. We calculated these probabilities for different boost values $b_s \in [-1, 1]$ for secure results using an increment of 0.1. Boost values b_{bp} and b_i remained the same. The results are shown in Figure 3.

Rank Trade-off— For IV, maximum probability p_{bp} for secure best-practices to appear in t_{10} and t_3 was given for $b_s = 0.1$ (see green line in Figure 3b). However, for KEY it was given for $b_s = -0.2$ (see the green line in Figure 3c). Therefore, those two values represented the upper and lower bound of b_s. To decide optimal $b_s \in [-0.2, 0.1]$ we searched for a point where the probability of secure results was higher than for insecure results. Figure 3a shows that $b_s > 0.0$ fulfilled this restriction for both tiers and all three tasks. Probabilities of insecure results were the lowest in all four plots. Therefore, we selected $b_s = 0.1$ as it was the next increment in the interval and the upper bound. This solution led to a small reduction of probabilities of secure best practices for KEY in t_{10} and t_3, as shown in Figure 3c and 3d. We considered this loss acceptable in order to keep insecure probabilities low. Further, this led to an additional gain for secure KEY results (see the blue line in Figure 3c and 3d).

Since results for CIPHER are only separated into a secure and insecure class, we did not have to include them in the parameter search. Secure CIPHER results obtained $b_s = 1.0$ and insecure results $b_i = -1.0$.

5.6 Results

This section provides the results for security best-practice clustering and the binomial distribution of boosted secure best-practice, secure, and insecure results in t_{10}.

Secure Best-Practice Clusters— DBSCAN identified 12 best-practice clusters for IV with overall 92 identified best-practice examples. The biggest cluster contained 16% of the samples, as shown in Table 3. We found 46 Stack Overflow webpages that were labeled secure by l_{t1} (the top answer provides only secure code examples) and contained at least one best-practice example in the top answer. For those results, we set the boost to $b_{bp} = 1.0$ in cse_b.

Ten best-practice clusters were found for KEY with overall 113 identified best-practice examples. The biggest cluster contained 17% of the samples. We found 11 webpages that were labeled secure by l_{t1} and provided at least one KEY best-practice example in the top answer. They also obtained a boost value of $b_{bp} = 1.0$ in cse_b.

Binomial Distribution— We computed the binomial distribution over $\{cse_b(q)\}_{q \in Q} \subset t_{10}$, where cse_b is the modified Google Search engine that applies our boosting values b_{bp}, b_s, and b_i. Q are the same queries we collected in Study 1. We calculated the binomial distribution for IV, KEY, and CIPHER by simulating 10,000 searches.

Task-Independent Results— We first measured the binomial distributions independently from the tasks. $B(10, 0.089)$ of boosted secure best-practice results revealed a probability of 38.16% for one result to appear in t_{10} and 29.36% in t_3. Both distributions over t_3 and t_{10} were even higher than the observed distributions of insecure results in Study 1 (see Figure 2). Further, we saw a 17.11% chance for two, and 3.85% for three results to appear in t_{10}.

$B(10, 0.061)$ of secure results showed a chance of 36.54% for one result, 11.59% for two, and 2.28% for three results to appear in t_{10}. As expected ($b_s < b_{bp}$) is lower than the distribution of secure best practices.

The distribution of insecure results $B(10, 0.008)$ was finally the lowest. The probability of one result in t_{10} was reduced from 36.46% (as observed in Study 1) to 5.57%. Moreover, we observed near-zero probability of an insecure result in t_3 which was 22.78% in Study 1.

IV— We measured the binomial distribution $B(10, 0.084)$ for boosted secure best-practice results $\{cse_{b_{bp}}(q)\}_{q \in Q_{iv}}$ over t_{10}. As shown in Figure 4, *(IV, t_{10}, green)* the probability of at least one result to appear in t_{10} was 38.96%, 18.70% for two results, and for three results, 6.08%. Comparing this distribution with the distribution of boosted secure results $\{cse_{b_s}(q)\}_{q \in Q_{iv}}$ also shown in Figure 2 (see IV, t_{10}, blue), we observed the required higher distribution of secure best-practice results.

Further, the distribution for boosted secure results shown in Figure 4, *(IV, t_{10}, blue)* was higher than the distribution of non-boosted secure results shown in Figure 2. The probability of at least one secure result in t_{10} increased from 22.44% to 25.03%, for two results from 2.86% to 3.52%, and from 0.19% to 0.3% for three results.

Lastly, we calculated the distribution $B(10, 0.010)$ of boosted insecure results $\{cse_{b_i}(q)\}_{q \in Q_{iv}}$, shown in Figure 4, *(IV, t_{10}, red)*, and compared it with the distribution of non-boosted insecure results from Figure 2. Probability decreased from 34.90% to a 9.32% for at least one insecure result in t_{10}, from 10.43% to 0.32% for two results, and from 1.65% to 0% for three results. Furthermore, there

was an increase from a 54.74% to a 90.36% chance that none of the results in t_{10} were insecure.

KEY— We see slightly different results for KEY due to the selection of b_s. It trades off higher probability of secure best-practice results to ensure that insecure results have lower probability than secure results across the board. This decision had an effect on the distributions for KEY.

$B(10, 0.047)$ of boosted secure best-practice results, as shown in Figure 4, *(KEY, t_{10}, green)*, shows a probability of 33.14% for at least one result, 8.99% for two results, and 1.4% for three results in t_{10}. $B(10, 0.094)$ of boosted secure results, shown in Figure 4, *(KEY, t_{10}, blue)*, revealed a higher probability of one secure result to appear in t_{10} with 37.47%, 22.52% for two, and 7.95% for three results. The distribution of boosted secure results slightly increased in comparison to non-boosted secure results shown in Figure 2.

The distribution $B(10, 0.007)$ shown in Figure 4, *(KEY, t_{10}, red)* for boosted insecure results $\{cse_{b_i}(q)\}_{q \in Q_{key}}$, largely decreased in comparison to non-boosted insecure results shown in Figure 2. For at least one result in t_{10}, probability decreased from 37.35% to 9.32%, and from 15.22% to 0% for two results. Further, there is a chance of 95.22% that none of the results are insecure, which was 43% for non-boosted secure results.

CIPHER— The distribution $B(10, 0.062)$ of boosted secure results $\{cse_{b_s}(q)\}_{q \in Q_{cip}}$ over t_{10}, shown in Figure 4, *(CIPHER, t_{10}, blue)*, is slightly higher than the respective distribution for IV and slightly lower than the one for KEY. The probability of at least one secure result in t_{10} increased from 21.06% to 36.76%, from 2.71% to 12.64% for two results, and from 0.24% to 2.19% for three, in comparison with non-boosted secure results from Figure 2. There is 0% chance that one of the results is insecure, which was 37.77% before, while the chances for no insecure results increased from 44.41% to 100%.

Summary— We have shown that the new distributions of boosted results show the required relative probabilities. Secure best-practice results have the highest distribution, secure results the second highest, and insecure results very low probabilities to appear in the top results. However, we observed one exception. Secure best-practice results for KEY actually have a lower distribution than secure results. This is due to the boosting trade-off that ensures that insecure results have the lowest probabilities across the board.

We have shown how clustering of secure best-practice results and boosting helps to perform security-based re-ranking. In the next step, we had to test whether the original ranking by *cse* and the re-ranking by cse_b actually had an effect on code security. In other words, would we observe different results with respect to code security if developers solved a programming task with the help of Google Search, while one group used *cse* and the other one cse_b?

6 STUDY 2: IMPACT OF RANKING ON CODE SECURITY

We performed an online study where participants were presented the same tasks from Study 1, i.e., CIPHER, IV, and KEY. This time they not only had to search for potential solutions online but also had to write small Java programs in order to solve them.

6.1 Setup

We used the online study framework Developer Observatory [43] to perform the study. It provides ready-to-use study templates, an online code editor, tracking functions, and security features.

Participants were first presented with a landing page that provided an overview of the study, followed by a consent form and an explanation about the online code editor and a Jupyter notebook that was given to solve the tasks. The notebook gave an introduction and overview of the tasks, followed by text/code cell pairs that described each task and provided code skeletons to solve them (see Figure 5). Each of the tasks could be run and solved independently from each other. To provide participants the opportunity to verify that all three tasks were solved functionally correct, the notebook provided a code cell to run and test the given solutions. The notebook applied a Java 8 kernel to execute code cells.

Participants were instructed to solve each task using the provided Google Search bar to look for help online. They were randomly assigned to one of our conditions. If assigned to the treatment group, the search bar was run by cse_b, which applied security-based re-ranking, as explained in Section 5. Participants that were assigned to the control group used *cse*, Google's default search engine. We provided the respective link to each search bar in the introduction section of the notebook. The search bar was opened in a new browser tab.

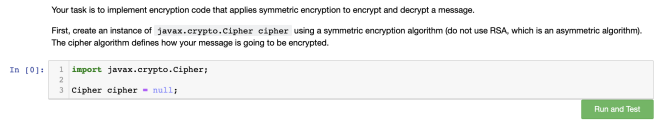
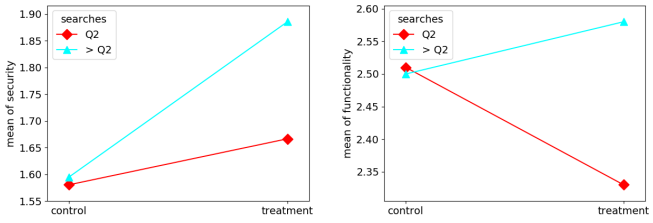


Figure 5: Example task from Study 2

6.2 Results



(a) Security **(b) Functional Correctness**

Figure 6: Interaction plots (number of searches above and below median Quantile Q2)

Search Depth— In order to evaluate whether our treatment helped in writing functional and secure code, we first investigated to what extent the search engine was actually used to solve the tasks. Participants may have not used the search engine at all, or performed very few searches for only a subset of tasks. In contrast, they may have used the search engine extensively.

	Security				Functionality			
	MS1	**MS2**	**MS3**	**MS4**	**MF1**	**MF2**	**MF3**	**MF4**
Condition: Treatment	0.083	0.256	0.004	0.062	**-0.485***	**-0.902*****	**-0.564***	**-1.251*****
	(0.181)	(0.210)	(0.210)	(0.244)	(0.214)	(0.238)	(0.256)	(0.283)
Condition × Searches†	-	-	-	**0.125***	-	-	-	**0.307*****
				(0.058)				(0.080)
Condition × Votes†			**0.004***				0.002	
			(0.001)				(0.002)	
Secure ∪ Best-Practice Results†	-	**0.106***	-	-	-	**0.296*****	-	-
		(0.052)				(0.076)		
No Searches Performed	-0.587	-	-	-	**-1.149****	-	-	-
	(0.373)				(0.333)			
Task: IV	**-1.254*****	**1.241*****	**-1.283*****	**-1.272*****	**-0.784****	**-0.767****	**-1.008*****	**-0.753****
	(0.223)	(0.221)	(0.0226)	(0.224)	(0.267)	(0.225)	(0.286)	(0.261)
Task: Key	**-1.325*****	**-1.312*****	**-1.246*****	**-1.343*****	**-0.575***	**-0.560***	**-0.521**	**-0.550***
	(0.223)	(0.223)	(0.226)	(0.225)	(0.271)	(0.268)	(0.303)	(0.265)

Table 2: Logistic regression results for SECURITY and FUNCTIONALITY. Significant values are highlighted in bold, and marked with: $^{+}$p<0.1, *p<0.05, **p<0.01, and ***p<0.001. Standard errors are included in parentheses. Variables marked with † are average counts per task.

More specifically, we expected an interaction effect between the number of searches and condition (i. e., control vs. treatment) on the security and functional correctness of submitted solutions. The number of searches was the average number of performed search queries per task. An interaction plot shows the relationship of the two observed variables *Searches* and *Condition*. We show the two relevant interaction plots in Figures 6a and 6b.

Figure 6a indicates the interaction effect between *Condition* and *Searches* on the mean of secure solutions (Max: 3 secure solutions per participant). It shows a red line which represents the median Quantile Q_2 of *Searches* and a blue line which represents *Searches* above Q_2. The left side of each line represents the results from the control group and the right side the results of the treatment group. Interaction plots indicate an interaction effect within a condition if their points (diamond and triangle) are far away from each other on the y-axis. This allows to easily spot differences between the conditions in the interaction effect if lines are not parallel.

We can easily observe that a higher number of searches has a different effect on security for each condition. In particular, the amount of searches did not have a substantial effect on control (i.e., the mean of secure solutions remained around 1.6 (blue and red). However, in the treatment group the mean of security increased from 1.6 (red) to 1.9 (blue) for search counts that are above the median. In other words, the more searches performed by participants in the treatment condition, the stronger the positive effect of the treatment on the number of securely solved tasks. We show later on that this interaction effect is statistically significant.

Figure 6b shows the interaction plot for functionality. We again observe that the amount of searches did not have an effect on control, since the mean of functional solutions remained around 2.5 (blue and red). In the treatment group we observed an increase of the mean of functionality from 2.35 (red) to 2.58 (blue) for search counts above the median. That means, the more the participants in the treatment group used the modified search engine the more functional solutions they submitted. We will show that this positive interaction effect is also statistically significant.

Figure 6b also indicates that the treatment group performed worse than control (2.51 vs. 2.34) for search counts below the median (red). We will show that this is likely caused by a difference in the subject pool, which we will explain later on. However, participants with high search activity in the treatment condition, outperform the participants in control irrespective of their degree of search activity.

The interaction plots indicate that the more that participants in the treatment condition used the search engine, the more secure and functional solutions they submitted. In the following, we provide several logistic regression models that show statistical significance of this key result.

Security— We provide four logistic regression models for security. We used Akaike information criterion (AIC) model selection to distinguish among a set of possible models describing the relationship between the condition, interaction effect, amount of searches, search results, tasks and background variables. We present the models MS1-MS4 which had the best fit in Table 2. Based on Peduzzi et al. (Rule of 10) and later results by Vittinghoff and McCulloch (suggesting 5-9 samples per independent variable) for logistic regression, our analysis should be adequately powered [33, 49].

MS1 shows that *No Searches Performed* had no significant effect on security for all participants, independently from the condition. This means participants that did not use the search engine at all showed no significant difference in the security of submitted solutions. MS2 shows that *Secure ∪ Best-Practice Results*, the average count of results per task that where secure or secure best-practices, had a significant positive effect on all participants ($p < 0.05$). The more the participants received secure or secure best-practice results, the more secure solutions they submitted. This already indicates that searching and receiving good results leads to better security. Therefore, we would expect participants from the treatment group

to perform better than the control group with increasing searches, since they are the ones expected to receive more secure and secure best-practice results. To measure this we include the interaction variable *Condition* × *Searches* in model MS4. It confirms our expectation: in the treatment group increasing searches had a significant positive effect on security ($p < 0.05$).

Figure 7 further explains this result by showing the distribution of received results and visited results across both conditions. The treatment group received more secure (45.8% vs 30.8%) and drastically more secure best-practice results (36.9% vs. 0.4%) than the control group; and also much fewer insecure results (17.3% vs. 68.8%). When studying click behavior, participants in the treatment condition visited more secure (41.0% vs. 15.4%) and secure best-practice results (25.6% vs. 1.3%). The control group predominantly visited insecure results with 84.3% of made clicks.

The two tasks IV and KEY had a significant negative effect on security ($p < 0.001$). This goes in line with our initial assumption that secure solutions are more difficult to achieve for these two tasks.

We calculated further models which included background variables, i.e., whether participants had a security background, were professional developers, and whether developing in Java was their primary job. None of those variables had a significant effect on security. This means that the treatment helped in improving security independent of whether or not participants had any distinguishing characteristics.

Functional Correctness— We again conducted our analysis for three models MF1-MF4 and made similar observations. We present the three best fits according to (AIC) model selection in Table 2.

MF1 shows that not searching at all had a significant negative effect on functionality ($p < 0.01$) for all participants independently from the condition. Further, MF2 shows that the average count of secure or secure best-practice results per task has a significant positive effect on functionality ($p < 0.001$). Further, once more, in the interaction of participants with the search engine, we find a robust significant positive interaction effect on functionality ($p < 0.001$) in the treatment group, as shown by MF4. The more the treatment group used the modified search engine, the more functional solutions they submitted. We calculated further models that included background variables. None of them had a significant effect on functionality.

Given our observations in Figure 6b, it is unsurprising that we find a residual negative baseline effect of being in the treatment group (*Condition: Treatment*) on functionality ($p < 0.001$), once the positive interaction effect is filtered out.

We performed further analysis to explain this observation and found an imbalance between conditions in the amount of participants that submitted only one functional solution (treatment: 19 vs. control: 9). We further observed that this imbalance is resembled in the average amount of searches performed by those participants (treatment: 6.32 vs. control: 10.44). While participants from the control group that submitted only one functional solution had a similar average amount of searches as performed by all remaining participants that submitted more than one functional solution (10.02), participants from the treatment group that submitted only one functional solution were way below average.

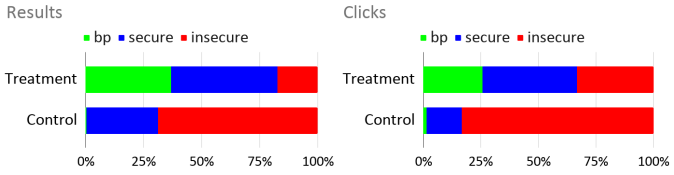


Figure 7: Results received and clicked

This low-effort search behavior can be caused by participants rushing through the study, solving the tasks on their own without searching, or side-using other sources (such as books, formal documentation, their own code repositories or other search engines). The treatment group had more participants who searched less often than the control group. Similar to the participants, who did not search at all, they did less well.

Note that the bottom line is that interaction with the treatment significantly improves security *and* functionality in comparison with the control group. This means with increasing search depth in both condition groups, the treatment group significantly and increasingly outperforms the control in submitting secure and functional solutions.

7 STACK OVERFLOW SIGNALS

We analyzed whether Stack Overflow's voting signals were reflected by the ranking of search results. Since we have shown that developers mostly click on one of the top three results it is important to evaluate whether those links provide posts with relatively low or high votes as it may influence the developer's decisions.

In order to measure to what extent the ranking of search results reflects Stack Overflow's voting signals, we calculated the Normative Discounted Cumulative Gain (NDCG) of search results from both condition groups in Study 2. NDCG is the standard metric used to evaluate information retrieval systems and reports a value between 0.0 and 1.0. It reveals how close a given ranking is to a defined ideal ranking. In our case, the ideal ranking r_V was created by ordering all Stack Overflow results for each search query in Study 2 based on the score of the top voted answer. A NDCG value of 1.0 means that the ranking perfectly matches the voting signals on Stack Overflow.

7.1 Reflection in Ranking

We obtained NDCG values based on the top ten results for each search query submitted during Study 2. We calculated two average NDCG values, one over all queries from the control group, and one for the treatment group. However, comparison between conditions has rather informative character since the set of queries of each condition is different, even though participants in both treatments solved the same tasks.

Ranking had an average NDCG of 0.89 in the control group and 0.85 in the treatment group. The obtained values reveal that the higher the rank of a result shown to our participants, the higher the vote of the top answer on the Stack Overflow post linked by the result. There was a much lower chance that results that lead to highly voted answers appeared on lower ranks.

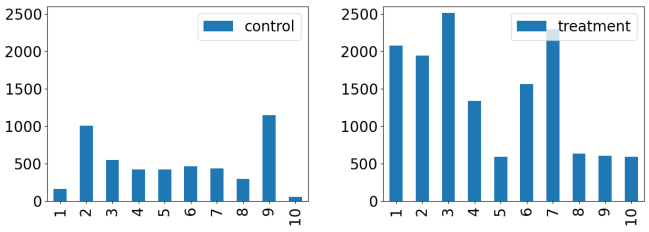


Figure 8: Sum of votes for secure results per rank

This is an important observation. If developers would frequently encounter lower-scored answers when clicking on top results, they may start looking for better scored content on the Stack Overflow site itself without using Google Search. This would subvert our intervention of security-based re-ranking. However, this risk is mitigated since top ranks lead to top voted content.

We further compared votes of top ranked secure and secure best-practice results in the treatment group with the control group. We observed that the sum of votes per rank was much higher in the treatment group on all ranks (except rank 9) and especially in the top three ranks as shown in Figure 8.[6]

We also included the voting score in regression models MS3 and MF3 (see Table 2). The interaction variable *Condition* × *Votes* measures whether higher voting scores in search results under treatment lead to better security or functionality outcomes. The variable was only significant for security ($p < 0.05$) but with a small effect size (0.004). As such, the effect of votes is clearly dominated by the number of searches (*Condition* × *Searches*).

Taken together, these findings are promising because for secure content our re-ranking intervention is compatible with Stack Overflow's own content evaluation system to a large extent. Therefore, developers can follow two signals they are used to pay attention to —Google rank and Stack Overflow votes—while being much better protected from inadvertently selecting insecure content.

7.2 Voting-Based Re-ranking

Similar to security-based re-ranking in Section 5, we calculated a boosting weight based on Stack Overflow's voting signals. This weight only considers the security label (i. e., secure/insecure) and the Stack Overflow score of the top voted answer. It does not know whether the top answer provides a secure best practice example (see Section 5.1). The advantage of *voting-based re-ranking* over security-based re-ranking is that it does not require manual labeling and clustering of secure best practices (see Section 5.3), as well as task-dependent ranking boosts (see Section 5.5).

Boost values for secure results were mapped to a value in $[0, 1]$ by applying min-max normalization on the Stack Overflow score. We did not consider the Stack Overflow scores for insecure results. Those still obtained a fixed negative boost of -1. We created a new search engine that applied those boosts and reapplied the search queries obtained from Study 2.

[6]The sum of votes per rank averaged over the number of results showed very similar results.

We compared the ranking of the results with Google's original ranking and security-based re-ranking using the same set of queries. We calculated two different NDCGs: $NDCG_S$ compares a given ranking with the ideal security ranking r_S, while $NDCG_V$ compares with ideal ranking r_V (as introduced in the beginning of Section 7). The ideal security ranking r_S ranks secure best-practice results before secure results, and insecure results are shown last.

Security-based re-ranking achieved the highest $NDCG_S$ with 0.89, while Google's original ranking had by far the lowest value with 0.45. Surprisingly, voting-based re-ranking was very close to security-based re-ranking with 0.86. With respect to $NDCG_V$, voting-based re-ranking was very close to Google's original ranking with 0.88 vs 0.89. Security-based re-ranking followed with 0.85.

The results show that voting-based re-ranking offers a security-scalability trade-off: it up-ranks secure results and down-ranks insecure results almost as good as security-based re-ranking and it up-ranks highly voted results very similar to original Google Search. Using this trade-off allows for a more scalable approach. It sidesteps manual labeling as well as clustering of secure best practices. Further, boosting values do not have to be customized to individual search tasks such as KEY or IV.

8 THREATS TO VALIDITY

Since we performed our studies online and not in a laboratory environment, we were not able to directly control side-use of other sources (i.e., books, formal documentation, own code repositories or different search engines).[7] On the one hand, running a study with actual developers at their place of (remote) work favorably adds to the realism and ecological validity of our work. However, this comes with a certain loss of control. Instead of solely using our search engine provided on the study webpage, participants may have opened a new browser tab and another search engine.

We were able to show that increase of interaction with *our* search engine (i. e., *Condition* × *Searches*) had a positive significant effect on security and functionality in the treatment group. In addition, receiving more secure and best practice results significantly improved outcomes for both treatment groups. Individuals in the treatment group disproportionally benefited from this direct effect on outcomes (see Figure 7). The question is if side-use may have effected any of these robust statistical results.

We consider two theoretical scenarios: one where side-use was significantly performed by only one of the condition groups and one where side-use was similarly distributed across conditions.

Regarding the first scenario, we observed a lower average search count for participants that submitted only one functional solution in the treatment group. This may have been caused by side-use. However, we do not think that the treatment itself would have caused potential side-use, since we did not observe smaller than average search counts for treatment participants that submitted more than one functional solution.

In the second scenario we assume side-use is equally likely performed by both groups. Here we would only expect differences if one group used a search engine different to Google Search significantly more often. For instance, the treatment group may have side-used Bing while the control group side-used Yahoo. However,

[7]Due to national and local COVID-19 restrictions, all studies were performed online.

we find this unlikely since all participants stated in the exit survey that they merely use Google Search to solve programming tasks.

If participants side-used Google Search, personalization may have let to different search results for individual participants. However, this effect would have evened out over the complete set of participants.

9 DISCUSSION AND FUTURE WORK

Our results show that Google Search ranking has a significant effect on code security. Therefore, we argue that Web mechanics can be leveraged in order to support developers to write secure code. It is a new approach in the research field of usable security for software developers and we will discuss the advantages and disadvantages of this approach in the following.

Scalability— While security-based re-ranking showed very promising results in our study it is still an open question whether it can be applied to a greater ecosystem consisting of different open source websites and multiple programming languages. Especially since our approach involves manual inspection and reliance on a pre-annotated dataset.

However, large-scale scanning of open-source code websites is already taking place. The code analysis platform LGTM[8] scans public repositories on Bitbucket, GitHub, and GitLab. It supports popular programming languages like C,C++, C#, Go, Java, JavaScript, and Python. It relies on deep learning to detect a wide range of CWEs for each of the supported programming languages. Additional invariant learning allows detection of zero-days.

Further, LGTM's code analysis tools[9] and results are publicly available and could directly be used by Google Search as a signal to update their ranking.

As we have shown in Section 7.1, Google Search's ranking very closely reflects the perceived relevance of results to coding problems. As such, Google may initially keep the relative ordering of their ranking and merely use the security label, e. g., a found CWE by LGTM or CodeQL, to down-rank those specific results. This solution may be "good enough" in the sense that it keeps developers away from the most dangerous CWEs.[10] As discussed in Section 7.2, this would eliminate the necessity of manual code inspection as well as customized ranking boosts for individual tasks (see Section 5).

Note that by re-ranking specific pages of a single prominent website, we were already able to demonstrate significant positive effects on code security. Therefore, we think that it is not necessary to scan and vet each and every webpage, but rather advocate for a smaller scale that considers hot-spots, i. e., frequently visited webpages. Google already has the necessary data including page rank and click logs to identify those webpages.

Transparency— Being a major player, Google already took advantage of their central position in order to successfully enforce security standards (e.g., HTTPS, CT). These standards can be enforced transparently with near-zero errors. However, identifying insecure code examples is not transparent to the user and may be associated with a non-negligible amount of false positives. A wrongly identified vulnerability may then unjustly down-rank a specific webpage. Note that our approach does not penalize the complete website domain but merely the single webpage that provides insecure code. It requires further work on how to design a protocol between website owners and search providers to communicate found vulnerabilities, required actions with a given timeline, and penalties. The problem is very similar to Google's approach on down-ranking webpages that provide misinformation. Those webpages obtain the lowest rating by their Search Quality Rater. However, it is unclear how accurate their ratings are.[11]

Human Factors— Our intervention remains completely invisible and does not require anything from the user. Developers do not have to be aware of it in order to use it [51]. They do not need to download, install, and learn how to use it [31]. They do not have to pay attention to understand and follow security warnings, indicators, or recommendations [29]. They do not need to evaluate whether vulnerabilities reported by code analysis tools are false positives. There are no disruptive effects on the main programming task [11]. Developers do not have to cope with incomplete or unhelpful documentation or gain advanced skills that are sometimes required to use security tools [51]. Therefore, typical factors that need to be addressed in the field of usable security tooling, such as unawareness, usability, habituation, and inertia may not have any negative effects on security in our approach. Developers can simply perform their default code search behavior.

Incentives— In previous work we have shown that roughly 15% of apps from Google Play contain vulnerable code due to reuse from information sources found on the Web [17]. Fahl et al. have shown that many pose serious privacy risks [15]. In light of Google's privacy-first initiative, a further stride into that direction by helping developers protect people's privacy constitutes a strong incentive. Moreover, security-based re-ranking provides a very powerful tool to incentivize website owners to fix provided vulnerable code. If the code was not fixed after a given time frame, the webpage is down-ranked. This could have a ripple effect, improving the security of open source code Internet-wide.

10 CONCLUSION

Web search is the default method for developers that are looking for help online to solve programming tasks. Our study shows that ranking of search results has a direct effect on the security of their final solutions. Up-ranking secure results while down-ranking insecure ones leads to significantly more secure programs. Likewise, up-ranking insecure results and down-ranking secure ones leads to more insecure programs. Unfortunately, Google Search currently inadvertently follows the latter practice. We have shown that probability for an insecure result to appear in the top three is 22.78%, while only 9.19% for secure ones. We provide a semi-supervised clustering approach that identifies webpages that provide secure best practices. Further, we have provided a parameter search that identifies effective ranking boosts for secure and insecure results. We integrated both mechanisms in Google Search and performed an online developer study. In this study, we demonstrate that the mechanisms we have generated significantly help developers to write secure code.

[8] https://lgtm.com/
[9] https://github.com/github/codeql
[10] See for example: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.
[11] https://static.googleusercontent.com/media/guidelines.raterhub.com/en//searchqualityevaluatorguidelines.pdf

Acknowledgements: We thank the participants of the reported studies, and the reviewers for their valuable comments.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *IEEE Symposium on Security and Privacy (S&P)*, pages 154–171, 2017.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *IEEE Symposium on Security and Privacy (S&P)*, pages 289–305, 2016.

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

[4] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle Mazurek, and Sascha Fahl. Security developer studies with Github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS)*, pages 81–95, 2017.

[5] Ricardo Baeza-Yates. Web usage mining in search engines. In *Web Mining: Applications and Techniques*, pages 307–321. IGI Global, 2005.

[6] Wei Bai, Omer Akgul, and Michelle L Mazurek. A qualitative investigation of insecure code propagation from online forums. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 34–48. IEEE, 2019.

[7] Hudson Borges and Marco Tulio Valente. What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[8] Simon Byers, Lorrie Faith Cranor, Dave Kormann, and Patrick McDaniel. Searching for privacy: Design and implementation of a P3P-enabled search engine. In *International Workshop on Privacy Enhancing Technologies (PETS)*, pages 314–328. Springer, 2004.

[9] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. How reliable is the crowdsourced knowledge of security implementation? In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 536–547, 2019.

[10] Mario Christ, Ramayya Krishnan, Daniel Nagin, Robert Kraut, and Oliver Gunther. Trajectories of individual WWW usage: Implications for electronic commerce. In *34th Annual Hawaii International Conference on System Sciences (HICSS)*, 2001.

[11] Anastasia Danilova, Alena Naiakshina, and Matthew Smith. One size does not fit all: A grounded theory and online survey study of developer preferences for security warning types. In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 136–148, 2020.

[12] Babur De los Santos. Consumer search on the internet. *International Journal of Industrial Organization*, 58:66–105, 2018.

[13] Rachna Dhamija, J Doug Tygar, and Marti Hearst. Why Phishing works. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 581–590, 2006.

[14] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: An empirical study of the effectiveness of web browser Phishing warnings. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074, 2008.

[15] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: an analysis of android ssl (in)security. In *2012 ACM Conference on Computer & Communications Security*, CCS '12, pages 50–61, New York, NY, USA, 2012. ACM.

[16] Adrienne Porter Felt, Robert W. Reeder, Alex Ainslie, Helen Harris, Max Walker, Christopher Thompson, Mustafa Embre Acer, Elisabeth Morant, and Sunny Consolvo. Rethinking connection security indicators. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*, pages 1–14, 2016.

[17] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? The impact of copy&paste on Android application security. In *IEEE Symposium on Security and Privacy (S&P)*, pages 121–136, 2017.

[18] Felix Fischer, Huang Xiao, Ching-Yu Kao, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley, Nat Buckley, Konstantin Böttinger, Paul Muntean, and Jens Grossklags. Stack overflow considered helpful! Deep learning security nudges towards stronger cryptography. In *28th USENIX Security Symposium (USENIX Security)*, pages 339–356, 2019.

[19] Stuart Geiger. Summary analysis of the 2017 GitHub open source survey. *arXiv preprint arXiv:1706.02777*, 2017.

[20] GitHub. Open source survey, 2017. Available at: https://opensourcesurvey.org/2017/. Last accessed on: June 13, 2020.

[21] Google. Transparency report: Google safe browsing, 2020. Available at: https://transparencyreport.google.com/safe-browsing/overview. Last accessed on: May 04, 2020.

[22] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS)*, pages 265–281, 2018.

[23] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 358–368. IEEE, 2015.

[24] Raman Goyal, Gabriel Ferreira, Christian Kästner, and James Herbsleb. Identifying unusual commits on GitHub. *Journal of Software: Evolution and Process*, 30(1):Article No. e1893, 2018.

[25] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. Measuring personalization of web search. In *22nd International Conference on World Wide Web (WWW)*, pages 527–538, 2013.

[26] Amir Herzberg and Ahmad Jbara. Security and identification indicators for browsers against spoofing and Phishing attacks. *ACM Transactions on Internet Technology*, 8(4):1–36, 2008.

[27] Michael Hucka and Matthew Graham. Software search is not a science, even among scientists: A survey of how scientists and engineers find software. *Journal of Systems and Software*, 141:171 – 191, 2018.

[28] Jing Jiang, David Lo, Jiahuan He, Xin Xia, Pavneet Singh Kochhar, and Li Zhang. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering*, 22(1):547–578, 2017.

[29] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *ACM/IEEE International Conference on Software Engineering (ICSE)*, pages 672–681, 2013.

[30] Tyler Moore, Nektarios Leontiadis, and Nicolas Christin. Fashion crimes: Trending-term exploitation on the web. In *ACM Conference on Computer and Communications Security (CCS)*, pages 455–466, 2011.

[31] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. A stitch in time: Supporting Android developers in writing secure code. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1065–1077, 2017.

[32] Bing Pan, Helene Hembrooke, Thorsten Joachims, Lori Lorigo, Geri Gay, and Laura Granka. In Google We Trust: Users' Decisions on Rank, Position, and Relevance. *Journal of Computer-Mediated Communication*, 12(3):801–823, 04 2007.

[33] Peter Peduzzi, John Concato, Elizabeth Kemper, Theodore Holford, and Alvan Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *Journal of Clinical Epidemiology*, 49(12):1373–1379, 1996.

[34] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. More common than you think: An in-depth study of casual contributors. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 112–123. IEEE, 2016.

[35] Md Masudur Rahman, Jed Barson, Sydney Paul, Joshua Kayani, Federico Andrés Lois, Sebastián Fernandez Quezada, Christopher Parnin, Kathryn T Stolee, and Baishakhi Ray. Evaluating how developers use general-purpose web-search for code retrieval. In *15th International Conference on Mining Software Repositories (MSR)*, pages 465–475, 2018.

[36] Miguel Ramos, Marco Tulio Valente, Ricardo Terra, and Gustavo Santos. AngularJS in the wild: A survey with 460 developers. In *7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 9–16, 2016.

[37] Ayushi Rastogi. Do biases related to geographical location influence work-related decisions in GitHub? In *IEEE/ACM International Conference on Software Engineering Companion (ICSE-C)*, pages 665–667, 2016.

[38] Elissa Redmiles, Amelia Malone, and Michelle Mazurek. I think they're trying to tell me something: Advice sources and selection for digital security. In *IEEE Symposium on Security and Privacy (S&P)*, pages 272–288, 2016.

[39] Caitlin Sadowski, Kathryn Stolee, and Sebastian Elbaum. How developers search for code: A case study. In *10th Joint Meeting on Foundations of Software Engineering*, pages 191–201, 2015.

[40] Yusuke Saito, Kenji Fujiwara, Hiroshi Igaki, Norihiro Yoshida, and Hajimu Iida. How do GitHub users feel with pull-based development? In *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 7–11. IEEE, 2016.

[41] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(1):1–25, 2011.

[42] Stack Overflow. Developer survey results, 2019. Available at: https://insights.stackoverflow.com/survey/2019. Last accessed on: May 04, 2020.

[43] Christian Stransky, Yasemin Acar, Duc Cuong Nguyen, Dominik Wermke, Doowon Kim, Elissa Redmiles, Michael Backes, Simson Garfinkel, Michelle L Mazurek, and Sascha Fahl. Lessons learned from using an online platform to conduct large-scale, online controlled security experiments with software developers. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2017.

[44] D. van der Linden, E. Williams, J. Hallett, and A. Rashid. The impact of surface features on choice of (in)secure answers by Stackoverflow readers. *IEEE Transactions on Software Engineering*, forthcoming.

[45] Gaurav Varshney, Manoj Misra, and Pradeep Atrey. A Phish detector using lightweight search features. *Computers & Security*, 62:213–228, 2016.

[46] Bogdan Vasilescu, Kelly Blincoe, Qi Xuan, Casey Casalnuovo, Daniela Damian, Premkumar Devanbu, and Vladimir Filkov. The sky is not the limit: Multitasking across GitHub projects. In *38th International Conference on Software Engineering*, pages 994–1005, 2016.

[47] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Perceptions of diversity on GitHub: A user survey. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 50–56, 2015.

[48] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. Gender and tenure diversity in GitHub teams. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 3789–3798, 2015.

[49] Eric Vittinghoff and Charles E McCulloch. Relaxing the rule of ten events per variable in logistic and cox regression. *American journal of epidemiology*, 165(6):710–718, 2007.

[50] Rick Wash and Emilee Rader. Too much knowledge? Security beliefs and protective behaviors among United States internet users. In *Eleventh Symposium on Usable Privacy and Security (SOUPS)*, pages 309–325, 2015.

[51] Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill. Technical and personal factors influencing developers' adoption of security tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers (SIW)*, pages 23–26, 2014.

[52] Yu Wu, Jessica Kropczynski, Patrick Shih, and John Carroll. Exploring the ecosystem of software developers on GitHub and other platforms. In *Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 265–268, 2014.

[53] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[54] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *ACM Conference on Computer & Communications Security (CCS)*, pages 363–376, 2017.

[55] Yanfang Ye, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, and Qi Xiong. ICSD: An automatic system for insecure code snippet detection in Stack Overflow over heterogeneous information network. In *34th Annual Computer Security Applications Conference (ACSAC)*, pages 542–552, 2018.

Appendix: Security-Based Re-ranking

```
SecureRandom rnd = new SecureRandom();
byte[] keyData = new byte[16];
byte[] iv = new byte[16];
rnd.nextBytes(keyData);
rnd.nextBytes(iv);
SecretKeySpec key = new SecretKeySpec(keyData, "AES");
```

Figure 9: Secure best-practice example for KEY

	Noise	Effort	Max	eps	minPts	Best-Practice
IV	0.114	0.07	0.155	0.04	2	12
KEY	0.092	0.045	0.172	0.03	2	10

Table 3: DBSCAN parameters for IV and KEY

Appendix: Study 2: Impact of Ranking on Code Security

	Nr. of Participants	Avg. Searches	Nr. of Tasks Completed			
			0	1	2	3
Treatment	118	8.47	13	8	5	92
Control	100	9.19	13	2	6	79

Table 4: Average searches and tasks completed