What Makes Security-Related Code Examples Different

Azadeh Mokhberi
University of British Columbia

Tiffany Quon
University of British Columbia

Konstantin Beznosov
University of British Columbia

Abstract

Developers relying on code examples (CEs) in software engineering can impact code security. We conducted semi-structured interviews with seven professional developers to investigate developers' habits, challenges, and strategies in the life cycle of using security-related code examples (SRCEs), with a focus on exploring the differences between security- and non-security-related CEs. Results indicate that a lack of adequately differentiating between SRCEs and non-security-related code examples (NSRCEs) is a reason for introducing vulnerabilities into the code. We found that developers had a habit of reusing vulnerable code from their previous projects. This code reuse unintentionally introduced the same vulnerability into new projects, while that vulnerability had already been fixed in later iterations of the original resource the CE had been taken from. Our results highlight that professional developers need the same number of such CEs even as they gain experience over time, while this may not be the case for NSRCEs.

1 Introduction

Though code examples (CEs) assist developers in learning and improving efficiency in software development, they can introduce vulnerabilities. According to Symantec Internet Security, 76% of websites scanned in 2016 had software vulnerabilities, with 9% of those vulnerabilities classified as critical [12]. Another report showed that around 97% of Java applications have at least one known vulnerability in one of their components [30]. The reuse of CEs may be a reason that contributes to the introduction of such security vulnerabilities, according to the recent findings that CEs in Q&A platforms may contain security vulnerabilities [1, 3, 9, 13]. A CE refers to code intended to be reused by developers via copying and pasting, albeit with various adaptations to the "borrowed" code. Developers use CEs for different tasks, including security, with the aim of seeking help [13], learning [6, 7, 17, 24, 26, 28], and improving development efficiency [34]. Also, CEs may give developers confidence about their code's security [18].

Due to the complex nature of security, there are, however, several caveats to consider about code security. Vulnerabilities in software have an enormous impact on its users [4]. Recent studies suggest that developers have difficulties with understanding and using CEs provided in security-related documents [2, 19, 21, 27, 32]. Also, one of the biggest challenges in learning APIs is a lack of proper examples of how to use them [16, 23, 24]. As a result, developers turn to CEs found on public forums and other sources that often include vulnerable code [2, 3, 5, 13]. Finally, reusing security-related code snippets necessitates caution and expertise [13]. While all of these studies emphasize the significance of taking precautions when using SRCEs, there is a lack of research investigating how these SRCEs differ from NSRCEs from the developer's perspective.

We explored the main differences between SRCEs and NSRCEs to better understand how the use of SRCEs propagates vulnerabilities in developers' code. The abovementioned findings encouraged us to investigate how developers differentiate between security-related and non-security-related CEs and if these two are different in terms of usage and harmful influence in developing secure software. Previous research in software engineering has identified qualities of good CEs [20, 29] in general. Specific to secure software engineering, researchers investigated why and how insecure code from Stack Overflow propagates to developers' code [5]. We built on these studies to investigate the following research questions:

RQ1: How are security-related and non-security-related CEs different from developers' point of view?

USENIX Symposium on Usable Privacy and Security (SOUPS) 2021.
August 8–10, 2021, Virtual Conference.

RQ2: What factors are linked to the value of CEs in the context of secure software engineering?

We define an SRCE as any code that contributes toward completing a task that is directly tied to security, such as encryption, authentication, authorization, hashes, certificates, SSL/TLS, etc. We conducted seven semi-structured interviews and used thematic analysis [8] to explore how SRCEs are different from non-security-related code examples (NSRCEs) for developers.

By building on previous research, this work makes two main contributions. First, we focused on comparing SRCEs and NSRCEs and our novel findings related to the main differences between SRCEs and NSRCEs. Second, we identified developers' challenges in the process of using SRCEs from the beginning to the end, the factors linked to the value of CEs, and developers' strategies to assess the security of SRCEs. Our results highlighted differentiating characteristics of SRCEs by demonstrating that professional developers require the same number of such CEs even as they gain experience over time, while this may not be the case for NSRCEs. In addition, our results showed that the stakes can be higher when using SRCEs, as developers' habits of using CEs from their previous projects can lead to the persistence of code vulnerabilities in software development.

2 Related Work

2.1 CEs in Software Engineering

While prior work has identified qualities of good CEs, these qualities do not guarantee a CE's correctness. Good CEs are concise and contextual, highlight important elements, provide step-by-step explanations, provide links to additional resources, and are ready to copy and paste directly into the developer's code [20, 29]. However, a trade-off between conciseness and robustness has been identified: while simple examples may be easier to understand, they may be less robust for different usage scenarios [25, 35].

Developers encounter challenges when reusing code, and some developers also modify CEs before using them. It has been found that developers seek out reusable CEs to improve personal efficiency [34] and that code reuse varies with team size and a developer's level of experience [1]. Xia et al. [34] discovered that finding reusable CEs online was challenging due to a lack of good search engine support, the difficulty of expressing intent in a search query, low levels of quality and reusability, and the difficulty of trusting CEs. Wu et al. [33] found that the majority of developers preferred to re-implement code rather than directly paste it from Stack Overflow, and that 31.5% of developers who used Stack Overflow code changed the code in some way before using it.

2.2 CEs in Developer-Centered Security

CEs play a critical role in secure coding. Using CEs may give developers confidence about the security of their code. Mindermann et al. [18] found that participants who used a CE to assist with their task were more confident in their solution's security. It has been found that factors contributing to secure API usage include useful examples in documentation [19], a focus on secure CEs in documentation [2], and contextual, actionable security advice [15].

Past studies have found that Stack Overflow CEs can contain security vulnerabilities. Acar et al. [3] found that the examples in API documentation, while harder to use, were more secure than those found on Stack Overflow. Other studies have also found insecure code on Stack Overflow [1, 3, 9, 13].

The number of views, comments, and favorites that Stack Overflow answers receive do not necessarily indicate their security. Answers containing insecure CEs were found to receive more views than secure answers [9, 35]. Chen et al. [9] found that insecure answers, on average, received "higher scores, more comments, [and] more favorites," and that 45% of "accepted" answers were insecure.

We designed our study based on the findings mentioned above. Perhaps the closest paper to our work is the study by Bai et al. [5]. The authors investigated the reasons why and how insecure code from Stack Overflow propagates to developers' code. With a broader approach, we considered all CEs resources, including Stack Overflow, to understand how SRCEs are different from NSRCEs, factors that affect developers' security-related decisions, and what developers need in SRCEs.

3 Methods

3.1 User Study Procedure

Beyond completing the qualification questionnaire and consent form, there were two main parts to the user study. First, prior to the interview, participants were asked to share two screenshots of SRCEs: one that they liked and one that they did not like (See Figures 1 and 2 for samples). The terms "liked" and "did not like" were intentionally left open-ended for the participants to interpret for themselves. Second, participants were asked to participate in an hour-long interview. During each interview, we asked participants about the two code samples they shared, what they liked or disliked about them, and how they used those CEs. Participants were then asked how they use SRCEs and NSRCEs in their work. We conducted semi-structured interviews to provide an opportunity for interviewees to share their thoughts with more freedom and in their own terms [11].

3.2 Data Collection and Participants

We used Reddit, Prolific, LinkedIn, and snowballing sampling to recruit participants. To be eligible to participate in this study, participants had to be 18 or older, currently live in the USA or Canada, have more than six months of professional software development experience (excluding internship/co-op experience), be currently employed in a software development role, have some professional experience developing code for cybersecurity-related tasks, and have experience using CEs for security-related tasks. After study adjustments based on the outcomes of two pilot studies, we recruited seven professional developers (illustrated in Table 1). All interviews were conducted using Zoom video calls. Participants interviewed in this study were compensated with $50 CAD. All data was collected during November and December of 2020. Before any data collection, this study was approved by the University of British Columbia's Behavioural Research Ethics Board (ID: H20-03253).

3.3 Data Analysis

The main researcher transcribed and two researchers coded audio recordings of all seven interview sessions, each around one hour long. Thematic analysis [8] was used for data analysis. Transcribed interviews were segmented and coded by two researchers to develop the code book. We followed the six phases of analysis outlined by Braun and Clarke [8].

4 Results

In this section, we present the findings that are related to how SRCEs are different from NSRCEs in forming the intent to find CEs, finding, assessing, and finally, implementing the CEs in the developer's code. Our results are organized according to the abovementioned phases were identified during analysis. In this study, we refer to participants by using the prefix "P" along with their ID. We use "CE" to refer to any type of code example, including SRCE and NSRCE.

4.1 Evolution of Usage Over Time

While the use of NSRCEs decreased or evolved as developers gained experience, the use of SRCEs remained similar. P2 and P5 (web application developers) used CEs more when they did not have much experience at the beginning of their careers. As they got more experienced, they tended to copy and paste NSRCEs less often than before. However, P2 noted that, for big projects, CEs were still needed. P2 and P5 still used SRCEs as often as when they started their professional career. This was because security was not part of their everyday tasks at work.

CE usage was also impacted by the increase in their perceived trustworthiness over time. While for some participants, such as P2 and P5, NSRCE usage decreased as they gained experience, for P3, a mobile application developer, CE usage increased in part from an increase in CEs' perceived trustworthiness: *"I can tell you nowadays [in comparison to the past] I use more examples because I think today [they have become] more available, more trustable. Yeah, I'm using more than [at the] time that [I] started programming."*

4.2 Reuse Challenges

Reusing SRCEs from previous projects had the potential to introduce vulnerabilities into code. One important difference between security- and non-security-related code had to do with reusing CEs. We found that, on one hand, participants used CEs from online resources. On the other hand, CEs could also be sourced from participants' own previous project(s) rather than the CE's original source. In this case, even if original security bugs were already fixed at the CE's source, the participants' copying and pasting of the old, vulnerable code from their own previous projects reintroduced the same vulnerabilities into their new projects. For instance, P3 explained his experience, observing: *"the example I sent to you, the in-app purchase [which had vulnerabilities, Figure 1], I used it in the five other applications. Yeah. And just did the same mistake . . . I used the code from the last project I wrote. I just copy-pasted them to other projects . . . , I did not just refer to the original documentation . . . . After a while, they changed their API and the documentation on the in-app purchases."*

Developers faced a lack of SRCEs in online resources, which impacted their ability to find CEs for usage and cross-reference purposes (Similar to the findings of previous research (e.g., [2, 19]). Participants mentioned that the number of available SRCEs was limited in comparison to the availability of NSRCEs. P2 stated that *"There are not that much resources as opposed to other area[s]. I can tell like in web development and front-end development, there are so many resources, but in security, [there] is not that much available . . . . I'm assuming it's because it's [security is] more difficult and involves mathematics and people they don't have the expertise to develop stuff like that."* Even when there were available SRCEs, there may not have been enough for cross-referencing or having multiple trustworthy options to choose from. Furthermore, one area that SRCEs were needed in was for secure coding practices that are new to the field of security. However, there were not enough CEs for new concepts (e.g., content security policy). P1 remarked: *"there is not enough information on the original website. You should search the net and most of the time, because it's new and maybe people are not facing this problem, we cannot find enough information, you cannot find your solution, even if you find something it's not secure! . . . There are no experienced people who can share their knowledge with others."*

4.3 Factors Linked to the Value of CEs

In this section we present our findings related to the factors that linked to the value of CEs to be selected for use. We found the value of a CE depends on its *content*, qualities of the CE itself, and its *context*, qualities beyond the CE's content that support and enhance the CE. We found that some factors were common to both SRCEs and NSRCEs, whereas other factors were specifically important for SRCEs.

Content

SRCEs should be understandable and comprehensive. While these qualities are important for all CEs, SRCEs without these qualities could lead to code vulnerabilities. Understandable CEs are written in such a way that, even without enough prior knowledge, developers can understand the CE. Three participants noted that SRCEs were more understandable when the code's functionality was broken down for explanation purposes. It was essential to understand the code to be able to find vulnerabilities in the CE. Otherwise, the code may introduce security bugs. For instance, P3 added a vulnerability to his source code because he did not understand the functionality of the code (discussed in Section 4.5). Next, SRCEs should also be comprehensive, meaning that the CE includes everything needed to be able to use the CE. This is important due to developers' lack of security knowledge. For instance, P5 explained that he encountered a CE shared in a Q&A platform with a missing symbol inside the code. The missing symbol made the CE vulnerable.

Context

As previous work shows that documentation of SRCEs is challenging for developers to understand [2, 3, 5, 15, 19], a similar finding in our study is discussed here.

Pop-up notifications on platforms containing SRCEs could warn developers to use the code with more caution. P5 shared one of his experiences where he received a warning to be more careful about evaluating the SRCE he were reviewing and said *"I think it might have been on Stack Overflow . . . or on GitHub . . . there was a little pop up that said 'You're looking at code that has a security vulnerability. So, don't copy and paste it.' "*

Specifically related to SRCEs in API documentation, it was critical to have a section that explained threats and risks. P3 noted that having a specific documentation section related to threats and risks was very helpful, and this section was preferably at the very beginning of such a document. P3 referred to the sample code he shared, which was from Google Play Developer API, and said: *"The Google Play [documentation] has a section for vulnerability and fraud. And I think it's really important . . . the other one [another CE shared with the researcher](Figure 1) I use doesn't have [those] things and I think that was the key point, which caused me that trouble. Because they didn't mention anything about 'you need to care about security!' "*

Material that aids in conceptual understanding was another critical requirement that should be included in the context of an SRCE. This is specifically important for SRCEs in using proper code to prevent attacks. The sample CE that P5 shared showed someone on a Q&A forum asking for a specific SRCE. P5 pointed out one of his shared CEs (Figure 4) and said: *"It doesn't just give a huge block of code, it kind of goes through and says, here's an encoded HTML because you need to actually get this information into HTML, you need to replace things to make it HTML, so that a cross-site scripting attack won't actually work, which is the end goal . . . . So, it's kind of like concept, concept, and then an example down here, which I think is nice."*

Another critical point, related to Q&A platforms, was the importance of reading comments related to the CE. Sometimes the CE was not complete and the author corrected their mistakes in the comment section, so that it was essential for developers to read those comments. In the sample code that P5 shared with the researcher, he explained that this sample might or might not be vulnerable. It all depended on whether the developer read the comment section.

4.4 Assessment Strategies

To assess the security of an SRCE, participants looked through the code, evaluated the security of the CE source, and relied on experts to assess the security of the SRCE. Main strategies included: (1) Trying to understand and compare the code to other similar CEs, which developed confidence about the security of the code. Developers gained confidence whenever they understood the CE and compared it to relevant API documentation. (2) Evaluating the quality of the code by seeing if it was clear, up-to-date, etc. (3) Assessing the source of the CE and selecting resources that they believed were more trustworthy. (4) Based on their own personal experience, deciding whether to trust the CE platform (e.g., Stack Overflow). (5) Evaluating the age of the CE based on the assumption that older CEs are more likely to be revised and fixed over time. (6) Relying on tests and reviews by colleagues before putting the code into the production environment. (7) Consulting with experts in their own organization or experts in an online forum. (8) Making decisions whether to trust obtained crowdsourced information while taking into account the aforementioned factors. Strategies 2, 4, 6, 7, and 8 are similar to [5].

Trusted experts were particularly helpful for junior developers and for those who had previously introduced vulnerabilities into their code. Participants sought help from experts for evaluating CEs specifically on two occasions: (1) at the beginning of their professional career, and (2) after experiencing a security breach by using a CE from an API documentation website. After this experience, consultations with expert developers via online forums became a new strategy for preventing vulnerabilities.

Security-related CEs were more scrutinized by developers. Participants were more lenient when searching for, assessing,

and using NSRCEs. For example, P3 said: *"[For a] problem which [is] not related to security, I prefer [to] just select the first thing I see because I want to just save time and I think it's not really [an] important task."* Compared to SRCEs, participants scrutinized NSRCEs less before copying and pasting the CEs. Additionally, reviews of security-related code were done with more care. Furthermore, trustworthiness of the CE source was more important in the case of SRCEs.

4.5 Challenges in Implementation

Understanding CEs

Challenges with understanding SRCEs can be categorized as challenges related to: lack of security-related knowledge, difficulties with understanding SRCEs' concepts, and missing or unclear context beside SRCEs. Previous work found that lack of security knowledge prevented some developers from evaluating and fixing vulnerabilities in SRCEs from online resources [5]). We also found evidence of this trend. The pre-existing knowledge the authors of CEs may assume that developers have, and security not being a part of a developer's day-to-day job were main reasons that made some challenges more significant for SRCEs. For instance, some SRCEs required pre-existing knowledge to understand the CE. P5 pointed out a specific part of the sample he shared and said: *"You could come in here and copy and paste this without knowing what a regex is like . . . it doesn't give any explanation of anything like that. I mean, at some point, that's assumed knowledge."*

Developers, even more experienced ones, had difficulties with understanding new security-related concepts in CEs. As security measures change over time, new security code concepts need to be explained. P2, who develops security-related code about half of the time for his job, remarked that there were difficulties with understanding new security concepts. He said: *"You might not be able even to get it [the concept of the CE] . . . . If I want to give an example, public and private key encryption is one of those things that is [a] really important concept in certificates and digital signatures. So, the basic idea, I can understand that. But there has been a new method called elliptic curve that has a new way of doing the public-key cryptography, but that was too difficult for me to understand."*

The information beyond the CE could be confusing without adequate explanation. Similar to the findings of previous research (e.g., [2, 3]), we also found that information around SRCEs is challenging. Extra information around an SRCE could be confusing when participants did not have a good enough understanding of security-related topics. Furthermore, a lack of documentation, improper documentation, or lack of security-related information (e.g., in API documentation) made it hard for developers to assess the vulnerability of CEs. For instance, P3 explained how the sample he shared led to a security vulnerability in his app. He explained his experience dealing with fraud caused by using a CE from API documentation and being misled by the documentation

Using CEs

A lack of security knowledge led some developers to not modify CEs and to naively rely on others. A lack of knowledge about how a CE worked caused developers to have difficulties modifying CEs for the task at hand. Some participants found security-related code hard to understand, they sometimes used and wrote code that they did not fully understand, as long as the code worked; the code would be considered "opaque." Also, a shortage of security-related knowledge led developers to rely on others' knowledge. P4: *"it's [security is] a very distant field that I don't deal with day-to-day, so I would take others' word for that than my own and others' experience ."* Similarly, P4 noted that if an SRCE would be complex due to lack of knowledge, she was less confident in making any changes to the code, and would instead use the code without any changes and rely on others' knowledge.

A shortage of security knowledge required developers to invest more time and effort to understand SRCEs. Using SRCEs was more time consuming for participants whose day-to-day work typically did not involve security-related code. As P2 noted: *"Encryption is not an easy concept that you can get in one day. You have to study that and see how it works in detail, you need to read some tutorials. . . . it's not really easy to comprehend."* Particularly with complex CEs, which need extra time and energy, developers gave up and turned to try other alternatives.

5 Discussion

Distinctions between SRCEs and NSRCEs are significant because they shed light on how developers' mindsets and behaviors in the use of SRCEs can increase the vulnerability of their code. In this section we discuss why knowing similarities and differences between SRCEs and NSRCEs is significant and how such knowledge could help future research.

5.1 General Discussion

Understanding differences between SRCEs and NSRCEs may help researchers and practitioners to better understand the needs of SRCE users. We found gaining more job experience does not necessarily reduce the need to use SRCEs, which means developers with different levels of experience still need and count on SRCEs, unlike NSRCEs. Such knowledge about the users of SRCEs would help the community to consider the needs of different users in the design of APIs, platforms, and educational materials. For instance, previous research suggested in-spot or nudging education [5, 14]. Our study would inform the design of such education by making clear the target user groups of such education. As a result, more informed decisions could be made about what level of explanation is needed in educational materials over time.

Another important outcome of our findings is to consider the critical role of awareness and education related to SRCEs. In terms of awareness, there is a need to raise developers' awareness around how to treat SRCEs differently. For instance, developers should be aware that they should not reuse SRCEs from their previous projects for two main reasons. First, the SRCEs they used may have vulnerabilities that were eventually fixed in the original sources of the SRCEs. Second, security measures change over time and developers need to use up-to-date secure coding practices. In this light, thinking about how we might raise developers' awareness about the differences between SRCEs and NSRCEs is another future avenue for this research.

Finally, developers face difficulties finding new and secure SRCEs for new security-related coding practices. A scarcity of SRCEs for new security-related coding practices may push developers to use insecure code or code that they do not fully understand. There are constant changes in the realm of security. As such, developers need understandable SRCEs (as discussed in 4.4) particularly for learning purposes when something new is introduced (discussed in 4.1 and identified by other researchers [6, 7, 17, 24, 26, 28]). Knowing this need could help practitioners, the makers of APIs, and online portal providers support developers when new security practices are introduced.

5.2 Implications for Future Research

Future research could shed light on how SRCEs could support the needs of developers with a wide range of job experience. As our participants gained more job experience, they needed fewer NSRCEs, similar to the findings of Abdalkareem et al. [1]. However, for SRCEs, we observed that increases in job experience did not appear to reduce SRCE usage. These findings could help to improve the design of platforms that house SRCEs. One promising avenue for future research could be to understand how online resources for SRCEs, such as Stack Overflow, can cater to inexperienced developers and experienced developers. Considering the job experience of developers seems important as it potentially affects how they see their abilities in general. However, in terms of security, they could be at the level of inexperienced developers. For instance, Bai et al.'s [5] findings imply that developers often use code without consulting external resources or formally verifying the code, in part because they are confident in their abilities. However, Bai et al. also identified a spread of insecure code from public forums to other code sources, which may indicate that this confidence does not guarantee code security. Therefore, more investigation is needed to better understand how to design platforms and SRCEs for developers with more job experience, especially when security is not part of their core, day-to-day work.

Mitigating a lack of knowledge requires on-the-spot training in online platforms. A shortage of security knowledge may prevent developers from the proper use of SRCEs. Similar to previous works (e.g., [10,31]), we found that when developers had less security knowledge, they tended to use more SRCEs, face difficulties in understanding security concepts in SRCEs, require more time and effort to understand SRCEs, be discouraged from modifying SRCEs, and naively trust the knowledge of online resource contributors, e.g., those who post on Stack Overflow. It does not seem feasible to expect developers to continuously update their security knowledge or to educate all developers. Instead, online platforms can create new features and policies to reduce the dissemination of vulnerable CEs from the start (e.g., applying nudges [14]). Further work is needed to investigate on-the-spot security training features that can help educate developers at the points where they need it. With the consideration of developers' different levels of knowledge and experience, on-the-spot learning features should be designed in a way such that various developers can use them. For instance, they can adjust the amount of extra information they require beside each CE based on their knowledge. In addition to creating mechanisms for linking posts on Stack Overflow to relevant educational materials, a measure suggested by Bai et al. [5], we suggest that information (e.g., about potential attacks) should be provided in CEs to warn developers about potential code vulnerabilities.

Reusing SRCEs from older projects has a higher risk of introducing vulnerabilities into code. Based on Rahman et al.'s [22] categorization of Stack Overflow comments, we found that comments related to bugs, errors, warnings, concerns, question clarification, code documentation, and tips and complementary information play an important role in helping developers identify vulnerabilities in SRCEs from such Q&A platforms. A critical point, related to Q&A platforms as shown in (4.4), is the importance of reading comments related to the CE. Additionally, we found that some developers have habits of reusing SRCEs from older source code, which might result in old code vulnerabilities "leaking" into the source code of new projects. One significant difference between SRCEs and NSRCEs had to do with reusing them. We found that when a developer reuses a vulnerable SRCE from an older project, they propagate vulnerabilities that may later be corrected in the original source.

A promising avenue for future research could be investigating ways to reduce the likelihood of developers reusing an SRCE from old source code that is now considered vulnerable. One option could be providing developers with new educational materials to teach them how to use online resources for SRCEs specifically, and what to consider when evaluating SRCEs. For example, developers could be provided with tool support for "linking" SRCEs in the project source code back to the online sources from which they were adapted. This way, those developers who are about to reuse an SRCE from an old project could easily find an updated version of the CE at the original source.

Acknowledgments

This research has been supported by the Huawei-UBC Software Engineering Technology Research Program. We would like to thank members of the Laboratory for Education and Research in Secure Systems Engineering, who provided their feedback on the reported research. Our anonymous reviewers provided important feedback and suggestions to improve the paper. Stylistic and copy editing by Eva van Emden helped to improve readability of this paper.

References

[1] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology*, 88:148–158, 2017.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson L. Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *IEEE Symposium on Security and Privacy*, pages 154–171, San Jose, CA, 2017. IEEE Computer Society.

[3] 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 *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, San Jose, CA, 2016. IEEE Press.

[4] 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. IEEE, 2016.

[5] 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, McLean, VA, 2019. IEEE Press.

[6] Ohad Barzilay, Amiram Yehudai, and Orit Hazzan. Developers attentiveness to example usage. In *Human Aspects of Software Engineering*, HAoSE '10, New York, NY, USA, 2010. Association for Computing Machinery.

[7] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, page 1589–1598, New York, NY, USA, 2009. Association for Computing Machinery.

[8] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[9] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. How reliable is the crowdsourced knowledge of security implementation? In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, page 536–547. IEEE Press, 2019.

[10] Sen Chen, Ting Su, Lingling Fan, Guozhu Meng, Minhui Xue, Yang Liu, and Lihua Xu. Are mobile banking apps secure? what can be improved? In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 797–802, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Deborah Cohen and Benjamin Crabtree. Qualitative research guidelines project, 2006.

[12] Symantec Corporation. Symantec internet security threat report. https://docs.broadcom.com/doc/istr-22-2017-en, publisher=Broadcom Inc. Accessed: 2021-05-03.

[13] 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 *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136. IEEE, 2017.

[14] 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 19)*, pages 339–356, Santa Clara, CA, August 2019. USENIX Association.

[15] 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 2018)*, pages 265–281, Baltimore, MD, August 2018. USENIX Association.

[16] Samuel G. McLellan, Alvin W. Roesler, Joseph T. Tempest, and Clay I. Spinuzzi. Building more usable apis. *IEEE Softw.*, 15(3):78–86, May 1998.

[17] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. How developers use api documentation: An observation study. *Commun. Des. Q. Rev*, 7(2):40–49, August 2019.

[18] Kai Mindermann, Philipp Keck, and Stefan Wagner. How usable are rust cryptography apis? In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 143–154, 2018.

[19] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 935–946, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming q a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34, 2012.

[21] Marten Oltrogge, Yasemin Acar, Sergej Dechand, Matthew Smith, and Sascha Fahl. To pin or not to pin—helping app developers bullet proof their TLS connections. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 239–254, Washington, D.C., August 2015. USENIX Association.

[22] Mohammad Masudur Rahman, Chanchal K Roy, and Iman Keivanloo. Recommending insightful comments for source code using crowdsourced knowledge. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 81–90. IEEE, 2015.

[23] Martin P. Robillard. What makes apis hard to learn? answers from developers. *IEEE Software*, 26(6):27–34, 2009.

[24] Martin P. Robillard and Robert Deline. A field study of api learning obstacles. *Empirical Softw. Engg.*, 16(6):703–732, December 2011.

[25] Mary Beth Rosson and John M. Carroll. The reuse of uses in smalltalk programming. *ACM Trans. Comput.-Hum. Interact.*, 3(3):219–253, September 1996.

[26] Forrest Shull, Filippo Lanubile, and Victor R Basili. Investigating reading techniques for object-oriented framework learning. *IEEE Transactions on Software Engineering*, 26(11):1101–1118, 2000.

[27] Justin Smith, Lisa Nguyen Quang Do, and Emerson Murphy-Hill. Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security. In *Sixteenth Symposium on Usable Privacy and Security ({SOUPS} 2020)*, pages 221–238, 2020.

[28] Jeffrey Stylos, Andrew Faulring, Zizhuang Yang, and Brad A Myers. Improving api documentation using api usage information. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 119–126. IEEE, 2009.

[29] Dirk van der Linden, Emma Williams, Joseph Hallett, and Awais Rashid. The impact of surface features on choice of (in)secure answers by stackoverflow readers. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[30] veracode. State of software security. https://www.veracode.com/sites/default/files\/Resources/Reports/state-of-software-securityvolume-7\-veracode-report.pdf, publisher=Veracode. Accessed: 2021-05-03.

[31] Daniel Votipka, Kelsey R. Fulton, James Parker, Matthew Hou, Michelle L. Mazurek, and Michael Hicks. Understanding security mistakes developers make: Qualitative analysis from build it, break it, fix it. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 109–126. USENIX Association, August 2020.

[32] Chamila Wijayarathna and Nalin A. G. Arachchilage. Why johnny can't store passwords securely? a usability evaluation of bouncycastle password hashing. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE'18, page 205–210, New York, NY, USA, 2018. Association for Computing Machinery.

[33] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Softw. Engg.*, 24(2):637–673, April 2019.

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

[35] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim. Are code examples on an online q a forum reliable?: A study of api misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896, 2018.

Appendices

	Gender	Country	Years of exp.	Type of software	Developing SRC	Type of job	Company size
P1	Female	Canada	10	Web Apps	Half the time	Full time	1,000 or more
P2	Male	Canada	5	Web Apps	Half the time	Full time	1,000 or more
P3	Male	Canada	6	Mobile Apps	Half the time	Freelancer	–
P4	Female	Canada	2	Web Apps	Sometimes	Full time	1,000 or more
P5	Male	US	2	Web Apps	Sometimes	Full time	1–249
P6	Male	Canada	3	Web Apps	Sometimes	Full time	250–999
P7	Male	Canada	4	Web Apps	Sometimes	Full time	1,000 or more

Table 1: Demographic information of all participants.

Figure 1: A code example that a participant did not like.

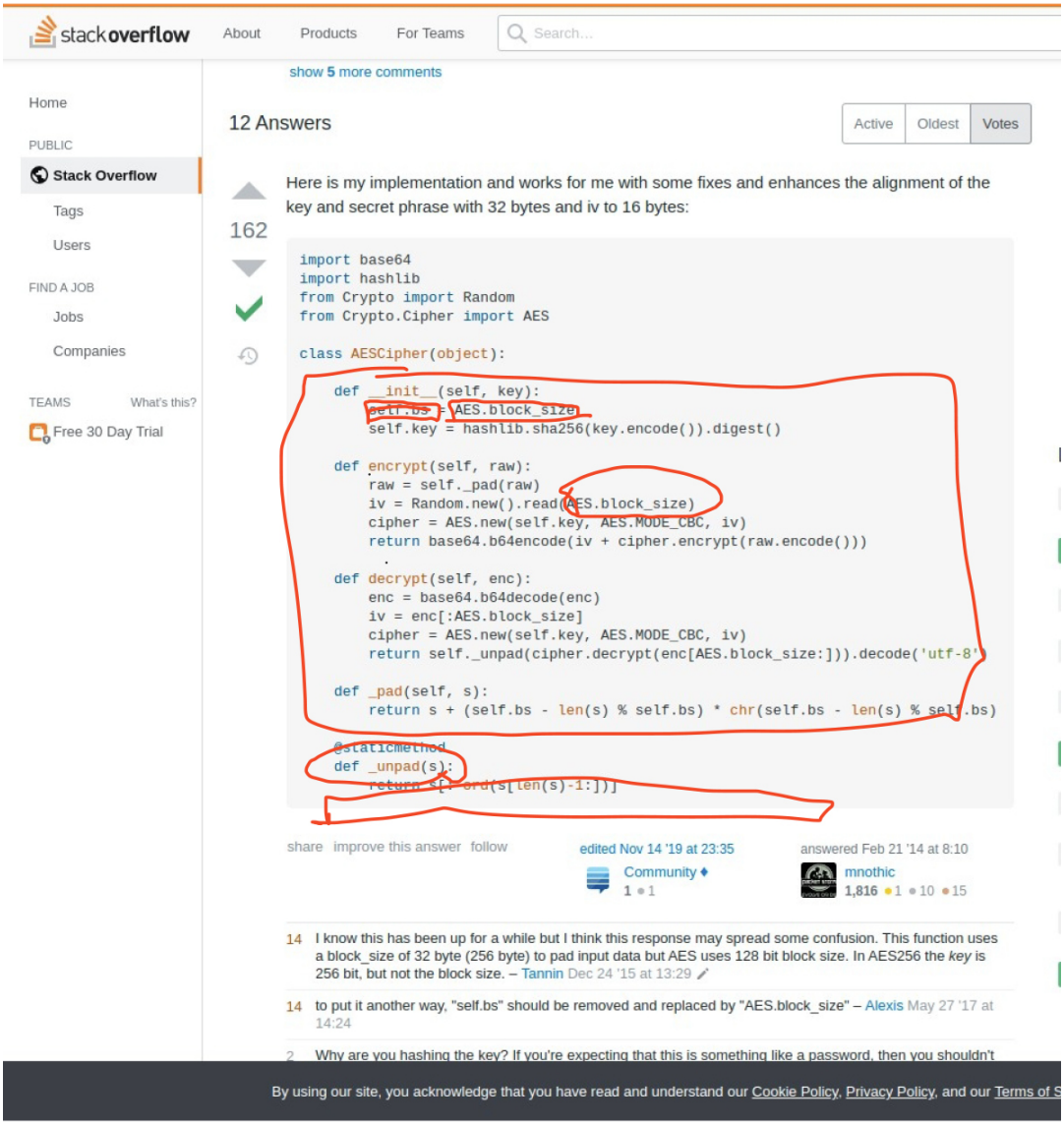

Figure 2: A code example that a participant liked.

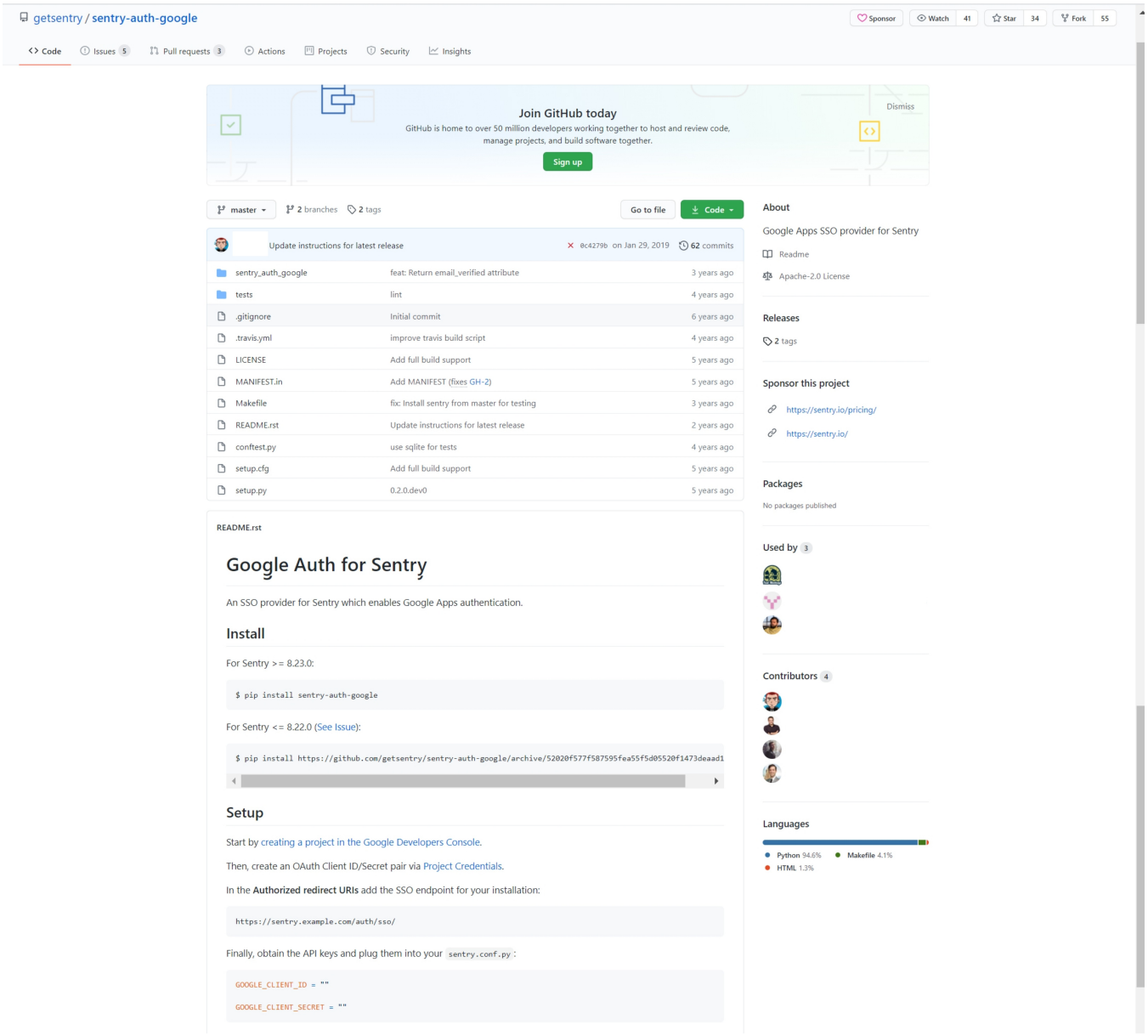

Figure 3: A code example that includes separated sections for explanation.

Never use `escape()`. It's nothing to do with HTML-encoding. It's more like URL-encoding, but it's not even properly that. It's a bizarre non-standard encoding available only in JavaScript.

If you want an HTML encoder, you'll have to write it yourself as JavaScript doesn't give you one. For example:

```
function encodeHTML(s) {
    return s.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/"/g, '&quot;');
}
```

However whilst this is enough to put your `user_id` in places like the `input value`, it's not enough for `id` because IDs can only use a limited selection of characters. (And `%` isn't among them, so `escape()` or even `encodeURIComponent()` is no good.)

You could invent your own encoding scheme to put any characters in an ID, for example:

```
function encodeID(s) {
    if (s==='') return '_';
    return s.replace(/[^a-zA-Z0-9.-]/g, function(match) {
        return '_'+match[0].charCodeAt(0).toString(16)+'_';
    });
}
```

But you've still got a problem if the same `user_id` occurs twice. And to be honest, the whole thing with throwing around HTML strings is usually a bad idea. Use DOM methods instead, and retain JavaScript references to each element, so you don't have to keep calling `getElementById`, or worrying about how arbitrary strings are inserted into IDs.

eg.:

```
function addChut(user_id) {
    var log= document.createElement('div');
    log.className= 'log';
    var textarea= document.createElement('textarea');
    var input= document.createElement('input');
    input.value= user_id;
    input.readonly= True;
    var button= document.createElement('input');
    button.type= 'button';
```

Figure 4: A code example that includes concept explanation.