A Study of C/C++ Code Weaknesses on Stack Overflow

Haoxiang Zhang, Shaowei Wang, Heng Li, Tse-Hsun Chen, and Ahmed E. Hassan, *Fellow, IEEE*

Abstract—Stack Overflow hosts millions of solutions that aim to solve developers' programming issues. In this crowdsourced question answering process, Stack Overflow becomes a code hosting website where developers actively share its code. However, code snippets on Stack Overflow may contain security vulnerabilities, and if shared carelessly, such snippets can introduce security problems in software systems. In this paper, we empirically study the prevalence of the *Common Weakness Enumeration* – CWE, in code snippets of C/C++ related answers. We explore the characteristics of $Code_w$, i.e., code snippets that have CWE instances, in terms of the types of weaknesses, the evolution of $Code_w$, and who contributed such code snippets. We find that: 1) 36 percent (i.e., 32 out of 89) CWE types are detected in $Code_w$ on Stack Overflow. Particularly, CWE-119, i.e., *improper restriction of operations within the bounds of a memory buffer*, is common in both answer code snippets and real-world software systems. Furthermore, the proportion of $Code_w$ doubled from 2008 to 2018 after normalizing by the total number of C/C++ snippets in each year. 2) In general, code revisions are associated with a reduction in the number of code weaknesses. However, the majority of $Code_w$ had weaknesses introduced in the first version of the code, and these $Code_w$ were never revised since then. Only 7.5 percent of users who contributed C/C++ code snippets posted or edited code with weaknesses. Users contributed less code with CWE weakness when they were more active (i.e., they either revised more code snippets or had a higher reputation). We also find that some users tended to have the same CWE type repeatedly in their various code snippets. Our empirical study provides insights to users who share code snippets on Stack Overflow so that they are aware of the potential security issues. To understand the community feedback about improving code weaknesses by answer revisions, we also conduct a qualitative study and find that 62.5 percent of our suggested revisions are adopted by the community. Stack Overflow can perform CWE scanning for all the code that is hosted on its platform. Further research is needed to improve the quality of the crowdsourced knowledge on Stack Overflow.

Index Terms—Code security, C/C++, empirical software engineering, crowdsourced knowledge sharing and management, stack overflow

1 Introduction

Stack Overflow is the world's most popular Q&A website for programming questions. Since its launch in 2008, Stack Overflow has accumulated millions of questions and answers related to programming. When answering questions on Stack Overflow, it is common for developers to attach code snippets within their answers as part of the solutions. Wu *et al.* observe that 75 percent of the answers contain code snippets [1]. The large collection of code snippets within these answers becomes a code repository for solving programming problems among developers. Prior studies show that the code snippets on Stack Overflow are widely shared by developers [1], [2].

Security is a critical property in any code repository. ISO 27005 defines vulnerability as a weakness that can be exploited [3]. Code with weaknesses – $Code_w$, can be risky to share or reuse among developers. As the world's most successful crowdsourced knowledge sharing platform in programming, Stack Overflow has hosted a very large code base. The activities of code sharing lead to code snippets propagating quickly across software systems. Prior studies observe that code snippets in various programming languages on Stack Overflow can be insecure. For example, Meng *et al.* identified security vulnerabilities, e.g., bypassing certificate validation and using insecure cryptographic hash functions, in the suggested code snippets of accepted answers on Stack Overflow [4]. Rahman *et al.* observed that 7.1 percent of the Python answers contain at least one insecure coding practice, e.g., code injection [5]. Fischer *et al.* observed that 15.4 percent of the 1.3 million Android applications contain security-related code snippets from Stack Overflow, and 97.9 percent of such code snippets contain at least one insecure code snippet [2]. Furthermore, on Meta Stack Overflow, which is the part of Stack Overflow where users discuss the inner workings and policies of Stack Overflow, we

- *Haoxiang Zhang is with the Centre for Software Excellence, Huawei Technologies Co Ltd Canada, Markham, ON L3R 5A4, Canada. E-mail: hzhang@cs.queensu.ca.*
- *Shaowei Wang is with the Department of Computer Science, University of Manitoba, Winnipeg, MB R3T 2N2, Canada. E-mail: shaowei@cs.umanitoba.ca.*
- *Heng Li is with the Department of Computer Engineering and Software Engineering, Polytechnique Montreal, Montreal, QC H3T 1J4, Canada. E-mail: heng.li@polymtl.ca.*
- *Tse-Hsun Chen is with the Software Performance, Analysis, and Reliability (SPEAR) Lab, Concordia University, Montreal, QC H3G 1M8, Canada. E-mail: peterc@encs.concordia.ca.*
- *Ahmed E. Hassan is with the Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, ON K7L 3N6, Canada. E-mail: ahmed@cs.queensu.ca.*

Manuscript received 1 Mar. 2020; revised 7 Feb. 2021; accepted 9 Feb. 2021. Date of publication 19 Feb. 2021; date of current version 18 July 2022. (Corresponding author: Shaowei Wang.) Recommended for acceptance by E. Murphy-Hill. Digital Object Identifier no. 10.1109/TSE.2021.3058985

observe that users are concerned about the vulnerable code that is shared on Stack Overflow.[1,2,3,4,5,6]

For instance, a user posted the following C code snippet in the first version of an answer[7]:

system("sudo rm —no-preserve-root -rf /");

This code snippet would wipe the entire hard drive. Within 4 minutes, another user removed the insecure code with a revision note saying "*some person may actually try your code without fully understanding it first*." In another example,[8] a user proposed an answer to print a message by initializing a variable string [100]. Within 5 minutes, another user commented that "*the biggest flaw here is the glaring security hole (buffer overrun!)*." More than two months later, the answerer revised the answer to fix the issue. However, the answer exposed a security vulnerability for more than two months.

In this paper, we focus on studying the weaknesses of C/C++ code snippets on Stack Overflow because C/C++ are widely used in different types of software systems [6]. In order to study code weaknesses, we use the *Common Weakness Enumeration* – CWE, a community-developed collection of common software security weaknesses.[9] C/C++ have the most reported CWE types of all the programming languages that contain CWE [7], [8], and have the most security vulnerabilities [9], [10]. In particular, our study aims to answer the following three research questions (RQs):

- RQ1: What are the types of code weaknesses that are detected in C/C++ code snippets on Stack Overflow?
- RQ2: How does code with weaknesses evolve through revisions?
- RQ3: What are the characteristics of the users who contributed to code with weaknesses?

In summary, this paper makes the following contributions:

- We scan 646,716 C/C++ code snippets from Stack Overflow answers. We observe that code weaknesses are detected in 2 percent of the C/C++ answers with code snippets; more specifically, there are 12,998 detected code weaknesses that fall into 36 percent (i.e., 32 out of 89) of all the existing C/C++ CWE types. Especially, we observe that CWE-119/416/190/476/415 are commonly detected in Stack Overflow as well as CVE instances in real-world software systems. We suggest that Stack Overflow can perform CWE scanning for all the code that is hosted on its platform.
- We analyze the trend of code weaknesses, and find that the proportion of $Code_w$ grew year by year doubling from 2008 to 2018 after normalizing by the total C/C++ code snippets that are posted in the corresponding years.
- We examine the code evolution history of all the posted C/C++ code snippets on Stack Overflow. We find that in general, code revisions are associated with a reduction in the number of code weaknesses. However, the majority of $Code_w$ have weaknesses in the first version of the code, and they are never revised.
- We conduct a study with users who contribute $Code_w$. We encourage Stack Overflow to improve the code review mechanism since users tended to commit the same weaknesses repeatedly. We observe that only 7.5 percent of users who posted C/C++ code contributed code weaknesses, and more active users contributed fewer weaknesses.
- Acar *et al.* analyzed how Stack Overflow threads are used by Android developers, and observed that developers can copy and paste insecure solutions [11]. Similarly, our study of mining C/C++ code snippets on Stack Overflow wishes to gain a better understanding of the impact of the Stack Overflow information source in terms of code security. The recommendations based on our findings can be used to improve the quality of Stack Overflow as an information source. The building of crowdsourced knowledge while managing any security risks can benefit Stack Overflow as an information resource provider, can benefit the software engineering community in sharing code, and can benefit developers in fixing their security issues.

Paper Organization. The rest of this paper is organized as follows. Section 2 introduces the background of code security on Stack Overflow. Section 3 describes our studied code snippets and our approach to detect weaknesses in these code snippets. Section 4 details the results from our case study. Section 5 discusses our findings and their implications. Section 6 discusses the potential threats to the validity of our findings. Section 7 surveys relevant work to our study. Finally, Section 8 concludes our study.

2 Background

2.1 Code Snippets and Their Security Weaknesses on Stack Overflow

Many software systems are written in C/C++, or rely on system components that have been written in C/C++. A survey of sourceforge.com in September 2004 notes that a substantial percentage of open source projects are using C (14,0 percent) and C++ (14,2 percent). By October 2019, the TIOBE index – an indicator of the popularity of programming languages, ranked C and C++ as No. 2 and 4, respectively [12]. Furthermore, out of the 839 CWE types, the programming language with the most reported CWE types is C/C++. There are 89 types of weaknesses, i.e., CWE types, that can be found in the C/C++ programming language. C contains 80 CWE types [7], and C++ contains 84 CWE types [8]. For comparison, Java contains 73 CWE types, and PHP contains 23 CWE types.

Therefore, in this empirical study, we wish to gain a deeper understanding of the weaknesses in code snippets on Stack Overflow by analyzing C/C++ code snippets – which is the programming language with the most CWE types out of all programming languages, within answers on Stack Overflow in RQ1. We refer to code snippets on Stack Overflow as the code snippets that are displayed within

1. https://meta.stackoverflow.com/q/318722/
2. https://meta.stackexchange.com/q/9460/
3. https://meta.stackoverflow.com/q/373629/
4. https://meta.stackoverflow.com/q/266180/
5. https://meta.stackoverflow.com/q/266339/
6. https://meta.stackoverflow.com/q/273058/
7. https://stackoverflow.com/revisions/35926150/1/
8. https://stackoverflow.com/posts/52633163/revisions
9. https://cwe.mitre.org/

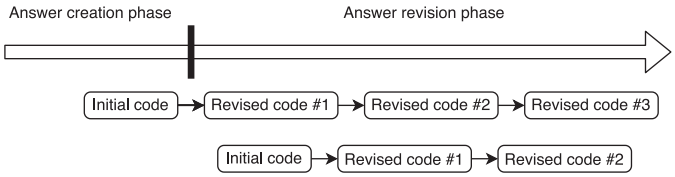

Fig. 1. Possible timelines for code snippet evolution during the answer creation and revision phases.

answers on Stack Overflow in the rest of the paper, if not otherwise specified. We offer actionable suggestions based on our empirical observations to improve the quality of the crowdsourced code knowledge on Stack Overflow.

2.2 The Evolution of Code Snippets on Stack Overflow

Stack Overflow encourages the community to revise the content of answers, including both textual description and code snippets, to maintain the quality of such answers. Any code snippet in answers can be revised, and new code snippets can be introduced to the answer at either the answer creation phase or revision phase. To illustrate the evolution of code snippets on Stack Overflow, we show both the answer evolution timeline and the two possible code snippet evolution timelines in Fig. 1.

Code weaknesses might be introduced at the creation of a code snippet or during the revision phase. Code weaknesses could also be removed during the revision phase. It is interesting to understand if such revisions help in improving the quality of code snippets in terms of their security weaknesses. Therefore, we study how $Code_w$ evolve through revisions in RQ2, e.g., when are the code weaknesses introduced, and whether the code revision helps in reducing code weaknesses? In addition, the activity level of contributors may have an impact on the quality of code snippets that they create or revise. For example, whether less active users are more likely to introduce weaknesses compared to more active users. Therefore, we wish to investigate the characteristics of contributors of code weaknesses and their relationship with code weaknesses in RQ3.

3 Study Design

Our study aims to gain a deep understanding of code weaknesses in Stack Overflow C/C++ answers. In this section, we first describe the process to create our datasets. Then we describe the motivation and approach of our study to answer the research questions. Fig. 2 illustrates the research approaches that we follow to conduct our study.

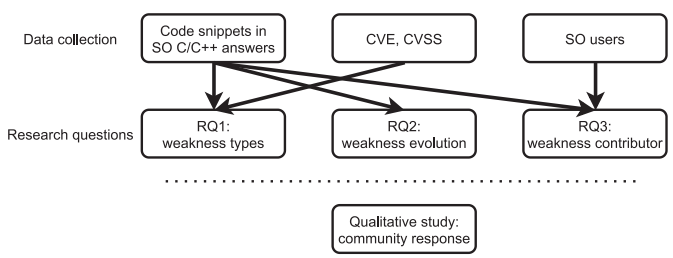

Fig. 2. Overview of our research approaches.

Fig. 3. An example of an accepted answer with a comment which flags the unsafe use of the string functions in the posted code snippet.

3.1 Data Collection

This subsection describes how we collect and construct our studied datasets. We first collect C/C++ code snippets from Stack Overflow answers. Then we detect code weaknesses in these collected code snippets by performing static code analysis. We elaborate on the details of each step below.

3.1.1 Collecting Code Snippets in Stack Overflow Answers

On Stack Overflow, developers post answers to specific questions that other developers ask. In addition to textual content, questions and answers may contain code snippets. Code snippets are segments of source code that are displayed with a gray background color embedded in the <code>...</code> HTML tags on Stack Overflow. Users can learn from such code snippets that are posted by others, and might even reuse such code snippets [13], [14], [15]. For example, in Fig. 3, a user posted a vulnerable C/C++ code snippet in a Stack Overflow answer.[10] The answerer posted the code snippet that used the string functions *strlcpy/strlcat*. A commenter pointed out that these functions were not safe and provided an alternative secure solution.

We analyze the SOTorrent dataset to study the weaknesses of shared code measured by the identified CWE types in such snippets. The SOTorrent dataset provides the version history of Stack Overflow at both the post level and the code snippet/text level within a post – question/answer [16]. Each code snippet has at least one version, i.e., the original version, or multiple versions, i.e., developers made revisions to the original version of the code. To understand security weaknesses in code snippets and their evolution on Stack Overflow, we leverage the version history of code snippets, i.e., code versions.

In this paper, we focus on studying code snippets in answers that are associated with the C/C++ tags because C/C++ are the languages that have the most security vulnerabilities [10]. More specifically, from the SOTorrent dataset[11] that is published in December 2018, we collect all the code snippets from answers that are associated with the C/C++ tags and their associated code versions. In SOTorrent, code version history is reconstructed by mapping a code

10. https://stackoverflow.com/a/48032218/
11. https://zenodo.org/record/2273117#.XZtTyEFKjmE

TABLE 1
The Statistics of the Studied Code Snippets With Weaknesses, i.e., $Code_w$

	Answer #	Code Snippet #	Code Version #
SOTorrent	867,734	1,561,550	1,833,449
LOC $>=5$	527,932	724,784	919,947
Guesslang	490,778	646,716	826,520
$Code_w$	11,235	11,748	14,934

block to its predecessors in prior post versions through syntax-based similarity metrics. As shown in Table 1, from the 1,598,646 answers in the C/C++ tags, we study the 867,734 (i.e., 54.3 percent) answers that have code snippets. From these answers, there are 1,561,550 code snippets, i.e., 14,194,563 lines of code in their latest version, with 1,833,449 code versions in total.

Many code snippets on Stack Overflow are pseudo code or command line functions, which may introduce bias to our study. To mitigate such bias, we further remove code snippets with less than five lines of code from the above-mentioned dataset by following prior studies [16], [17]. Baltes *et al.* observed that the median line count of code blocks on Stack Overflow is five [16]. We use the median line count (i.e., five lines of code) as a cutoff value to remove trivial code snippets and to ensure our studied code snippets are meaningful. As a result, we obtain 724,784 code snippets (with 919,947 code versions) from 527,932 answers. We also observe that code snippets in answers that are associated with the C/C++ tags are not necessarily C/C++ code. For example, users may tag a question as C/C++, but may put none-code text within the HTML <code>...</code> tags. Therefore, we use a tool called Guesslang[12] to determine whether a code snippet is actually written in C/C++. Guesslang generates a probability of a code snippet being one of 20 pre-defined programming languages, including C and C++, with a guessing accuracy higher than 90 percent according to its website.[13] We check the top five language guesses to see if any of them is C or C++ because we notice that this simple criterion gives high accuracy. We manually check the languages predicted by Gueslang using 100 randomly sampled code snippets, which ensure that we can reach a confidence level of 95 percent and a confidence interval of 10 percent. We find that the programming language of 91 (i.e., 91 percent) of these code snippets is correctly determined. By using Guesslang, we obtain 646,716 C/C++ code snippets (with 826,520 code versions) from 490,778 answers.

3.1.2 Detecting Weaknesses in Code Snippets

Common Weakness Enumeration, i.e., CWE, is a community-developed list of common software security weaknesses.[14] In order to detect C/C++ code snippets with weaknesses, i.e., $Code_w$, we use a static C/C++ code analysis tool called Cppcheck[15] to scan all of the 826,520 code versions of the resulting C/C++ code snippets. Cppcheck is a static analysis tool for C/C++ that supports various source code level checks, e.g., memory/resource leaks, automatic variable checking, and bounds checking [18], [19], [20]. It supports static checks that may not be covered by the compiler itself [21]. Cppcheck is widely used in error analysis for software systems, such as OpenOffice.org,[16] and Debian.[17] From prior studies, Cppcheck is observed to be highly precise. For example, Pomorova *et al.* observed that Cppcheck has a precision of 89 percent [22]. Arusoaie *et al.* observed that all the reported errors by Cppcheck were accurate in their experiment [23]. Note that Cppcheck can identify 59 out of the 89 types of code weaknesses in C/C++.[18]

The accuracy of Cppcheck is subject to a rigorous evaluation. Three raters, i.e., the first three authors, constructed an oracle dataset of 100 C/C++ code weaknesses – CWE instances, on Stack Overflow that were detected by Cppcheck. Each CWE instance was manually examined by at least two raters to determine whether it is a true CWE instance or not. We observe that Cppcheck achieves an accuracy of 0.85 – 85 out of the 100 detected CWE instances are labelled as true CWEs. Note that any disagreement was discussed until consensus was reached among the three raters. We also observe that the agreement among the raters is substantial with a Cohen's Kappa of 0.68 [24].

We observe that 682,588 code versions have no weakness – no CWE is reported for the scanned code snippets, and 143,932 code versions have at least one weakness, i.e., a CWE instance that is reported by Cppcheck). In addition, we collect all the 154,198 CWE instances. Among all the CWE instances, Cppcheck reports that 129,395 CWE instances are related to syntax errors. Such syntax errors are usually due to the incomplete nature of Stack Overflow code snippets. We remove these instances from our analysis and focus on the resulting 24,803 CWE instances in our following study, which come from 11,748 $Code_w$ in 11,235 answers. Note that a code snippet with weaknesses can be from either the latest version of an answer or the earlier versions. From these 11,748 $Code_w$, we collect 14,934 code versions with weaknesses, i.e., $Version_w$, out of the 17,591 code versions. Note that these 11,235 answers with code weaknesses – $Answer_w$, are associated with 10,634 questions, indicating that a question can have more than one answer with code weaknesses. In the rest of this paper, we further study these CWE instances and their evolution in Stack Overflow code snippets. In RQ1 (Section 4.1), we analyze $Code_w$ and their associated CWE instances from the latest version of answers in order to capture the current state of C/C++ security weaknesses from Stack Overflow. In RQ2 (Section 4.2) and RQ3 (Section 4.3), we analyze the evolution of these $Code_w$ in order to capture the evolution of C/C++ security weaknesses and the user aspects of such weaknesses.

3.2 Study Approach

This subsection discusses the approaches of our empirical study.

12. https://pypi.org/project/guesslang
13. https://github.com/yoeo/guesslang
14. https://cwe.mitre.org
15. http://cppcheck.sourceforge.net
16. https://wiki.documentfoundation.org/Development/Cppcheck
17. https://lwn.net/Articles/420252/
18. We examined the source code of the Cppcheck version that we used and observed that it supports 59 CWE types.

3.2.1 RQ1: What are the Types of Code Weaknesses That are Detected in C/C++ Code Snippets on Stack Overflow?

Motivation. Stack Overflow has a large number of C/C++ code snippets. Code with weaknesses can lead to security vulnerabilities if it is carelessly shared among developers. To gain first-hand insights of C/C++ security weaknesses on Stack Overflow, in this RQ, we analyze all the C/C++ code snippets in answers and investigate the characteristics of CWE instances that are detected in these code snippets. We wish to find out how commonly each CWE type is detected in the C/C++ code snippets on Stack Overflow, and the impact of these CWE types on real-world software systems. By answering this RQ, we wish to provide insights to developers on potential security risks when reusing Stack Overflow code snippets, and to inform Stack Overflow about potential security risks together with their trends over time, thus further action can be proposed to enhance the quality of crowdsourced code snippets on Stack Overflow.

Approach. To understand how common different types of security weaknesses are detected on Stack Overflow, we first analyze the CWE instances and their types as reported by Cppcheck in the code snippets of the latest versions of C/C++ answers. After identifying the types of CWE instances, we analyze the characteristics of different CWE types, e.g., the proportion of each CWE type among all CWE instances and the trend of different CWE types in terms of their number of instances over time, that is, from September 2008 to December 2018.

Furthermore, to evaluate the impact of our detected security weaknesses of each CWE type, we map each CWE type to its associated vulnerabilities in the *Common Vulnerabilities and Exposures* – CVE, which is a database containing vulnerabilities that are exposed in real-world software systems [25]. While a CWE instance represents common patterns of vulnerabilities in source code, a CVE instance represents actually vulnerable instances within real-world software products or systems. For example, CVE-2019-15916[19] is a denial of service overflow vulnerability reported in the Linux kernel. That CVE is associated with CWE-119,[20] i.e., *improper restriction of operations within the bounds of a memory buffer*. We wish to characterize the impact of different CWE types by examining the number of CVE instances that are related to each CWE type. A CWE type with a larger number of CVE instances indicates that this CWE type has a more practical impact in terms of security vulnerabilities on real-world software systems. We also use the median score of the *Common Vulnerability Scoring System* – CVSS,[21] of all the CVE instances within a CWE type to represent the severity of a CWE type. In this study, we are specifically interested in the security impact of each code weakness, i.e., CWE, and cvedetails.com enables us to count the number of CVE instances for each CWE type directly. On the other hand, nvd.nist.gov only lists CVE instances with their associated CWE types. Hence, we collect the vulnerability data from cvedetails.com. To evaluate the impact of different CWE types, we also refer to the 2019 CWE top 25 list that is made by the CWE team [26], [27]. The list ranks weaknesses based on both their prevalence and the severity of their associated CVEs. The list is a demonstration of the most widespread and critical weaknesses with potentially serious vulnerabilities. Weaknesses that are both popular and severe can rank high in the CWE top 25 list.

Lastly, to understand how answers with code weaknesses are recognized by the community, i.e., through vote or acceptance of an answer, we analyze the distribution of answer scores across the latest code snippets with different numbers of CWE instances, and calculate the Spearman's rank-order correlation to understand whether answers containing more weaknesses are less likely to be upvoted. We also calculate the proportion of answers with weaknesses that are accepted by the askers to understand whether answers can still contain weaknesses even though they are accepted.

3.2.2 RQ2: How Does Code With Weaknesses Evolve Through Revisions?

Motivation. Stack Overflow answers, including their associated code snippets, can be revised through revisions as an effort to maintain the quality of the crowdsourced knowledge [28]. Security weaknesses might be introduced or eliminated through the evolution of such code snippets. To gain a deeper understanding of when such $Code_w$ are introduced and how they evolve, we study the evolution of $Version_w$. We wish to provide insights into the impact of code revisions on the security weaknesses of code snippets, e.g., whether the revision mechanism helps improve the quality of code snippets in terms of their security weaknesses.

Approach. We first investigate when security weaknesses are introduced throughout the evolution of $Code_w$, i.e., which versions of code snippets start to contain weaknesses. To do so, we collect all the versions of C/C++ code snippets, i.e., including the initial and revised code, in Stack Overflow answers, and scan them with Cppcheck to identify CWE instances and their corresponding types. More specifically, we examine all code snippets that have ever been revised and analyze whether revisions help improve such snippets in terms of security weaknesses. We compare the first and last versions of a code snippet and aim to identify the patterns of the evolution, i.e., whether the last version of a code snippet has an additional, a reduced, or an equal number of weaknesses compared to the first version of a code snippet. Furthermore, we exam these patterns within different CWE types. We also analyze code revisions by comparing all the consecutive code versions to understand the evolution of code quality over time.

We define the code snippets whose last version has more CWE instances than their first version as *deteriorated* $Code_w$, while code snippets whose last version has less CWE instances than their first version as *improved* $Code_w$, and code snippets whose last version has the same number of CWE instances as their first version as *unchanged* $Code_w$. If a code snippet is identified as *improved* $Code_w$, we consider that revisions to this code snippet improve it in terms of reducing security weaknesses.

We observe that users can point out the security issues of a code snippet in comments. Therefore, it is also interesting to investigate the relationship between the number of

19. https://www.cvedetails.com/cve/CVE-2019-15916/
20. https://cwe.mitre.org/data/definitions/119.html
21. https://nvd.nist.gov/vuln-metrics/cvss

comments of an answer and the quality of $Code_w$, i.e., whether the associated answer of $Code_w$ that has more comments is more likely to be improved eventually.

3.2.3 RQ3: What are the Characteristics of the Users who Contributed to Code With Weaknesses?

Motivation. During the creation or revision of a code snippet, users may introduce CWE instances in the code snippet as shown in Section 4.2. In this RQ, we study the users who introduce CWE instances, i.e., by either posting or editing code, during the evolution of $Code_w$. Furthermore, we explore the characteristics when they contribute $Code_w$. A better understanding of such users who participate in the evolution of code snippets can provide insights for Stack Overflow to improve their current mechanism, e.g., code revision, for better code security practices.

Approach. We first identify those users who contributed $Version_w$. We examine whether the majority of $Version_w$ were contributed by a small group of the users. Next, we examine whether the activity level of a user on Stack Overflow is associated with the likelihood of their involvement in $Version_w$. To do so, we estimate a user's activity level through two aspects: the number of code versions that were performed by a user and the reputation of the user. It is challenging to measure a user's activity level on Stack Overflow. In our study, we use both code revision count in C/C++ posts and reputation points as proxies to measure user activity level. The reputation of a user is a common proxy to measure user activity level in prior studies [5], [29], [30], [31]. However, the reputation of a user is not directly related to his/her activity level on Stack Overflow. For example, a user can have a high reputation even if he/she only asks popular questions and never posts any answer. Therefore, reputation can be biased to the types of activities, i.e., asking or answering, or activities in other programming languages. To alleviate bias from reputation points, we also use the number of code revisions made by a user to measure their activity level in maintaining code snippets on Stack Overflow. Furthermore, we only consider the number of C/C++ code revisions, which directly reflects a user's activity level with regard to C/C++.

We analyze the correlation between the number of contributed code versions by a user, including both $Version_w$ and code versions without weaknesses, and the code weakness density, i.e., the proportion of $Version_w$, in all the posted code versions by the same user. Second, we study the characteristics of those involved users who contribute any code version, including both $Version_w$ and code versions without weaknesses, using their gained reputation points, as a proxy for user activity/involvement on Stack Overflow [29], [31]. We examine whether the number of CWE instances within the contributed $Version_w$ by a user is correlated with the gained reputation points by the same user. More specifically, we wish to examine whether or not active users in terms of reputation points are more likely to post secure code snippets. Finally, we wish to understand if users repeatedly contribute the same type of weaknesses, i.e., the same CWE type. We calculate the number of CWE instances by a user across different CWE types. For users who repeatedly post the same CWE type, we measure the timespan of the CWE instances between the first and last CWE instance.

Measurement of User Reputation. The Stack Overflow platform only provides the current reputation points of a user. In order to measure the reputation points of a user when he/she posts or edits a code version, we first crawled [22] the information of all users who posted/edited C/C++ code snippets, including $Code_w$ and code snippets without weaknesses, about their daily gain of reputation points. To calculate the reputation points when a user posted/edited a code snippet, we sum up the daily gain of reputation points before the date of the specific code version.

3.2.4 Qualitative Study: How Does the Stack Overflow Community Respond to Security Issues of C/C++ Code Snippets?

By the quantitative analyses described in Section 3.2.2, we aim to understand whether code revisions have an impact on the security weaknesses of Stack Overflow code snippets. To better explain the impact of code revisions in code weaknesses reduction [32], we conducted a qualitative study to find out the community feedback about the improvement to code weaknesses. More specifically, we randomly selected 40 $Answer_w$ from which Cppcheck detected code weaknesses and manually suggested revisions to the corresponding $Code_w$ to fix these CWE instances. Then we collected the feedback towards our suggested answer revisions, e.g., approving our suggested edit, or rejecting our suggestion to revise the answer.

4 Evaluation Results

This section provides the detailed results of our empirical study for analyzing the characteristics of code with weaknesses – $Code_w$, in terms of the types of weaknesses in Section 4.1 (RQ1), the evolution of $Code_w$ in Section 4.2 (RQ2), and the contributors of such code snippets in Section 4.3 (RQ3).

4.1 RQ1: What are the Types of Code Weaknesses That are Detected in C/C++ Code Snippets on Stack Overflow?

36 percent (i.e., 32 out of 89) of all the C/C++ CWE types are identified in the C/C++ code snippets on Stack Overflow. 12,998 CWE instances are identified within the latest versions of the 7,481 answers. Fig. 4 shows the number of CWE instances in each of the 32 CWE types. The definition of our detected CWE types can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2021.3058985. The top six most frequent CWE types are:

- *CWE-908, i.e., the use of an uninitialized resource, with 7,041 (54.2 percent) instances:*The resource is not properly initialized, and the program can change in an unintended way.
- *CWE-401, i.e., improper release of memory before removing last reference, with 1,820 (14 percent) instances:* Memory is not properly released, and an attacker may take advantage of the program in a low memory condition or even launch a denial of service attack.

22. We crawled the data on May 12, 2019.

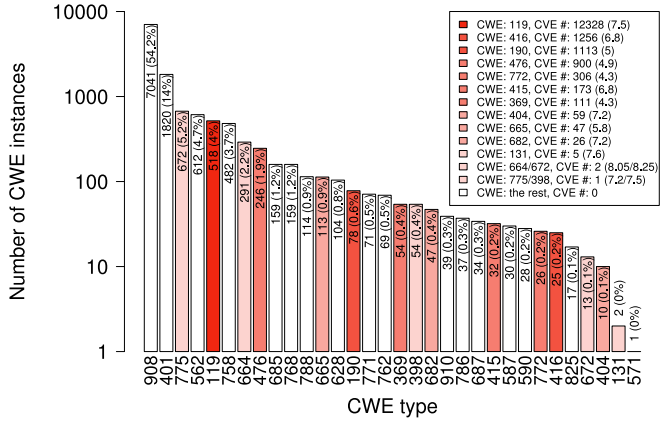

Fig. 4. The number of detected CWE instances in the latest versions of answers on Stack Overflow, and the number of CVE instances that are related to each CWE type. The intensity level of the color, e.g., red, indicates the frequency of the reported CVE instances in real-world software systems. A darker red indicates that more CVE instances were reported in real-world software systems. The median CVSS score of CVE instances in each CWE type is shown in parentheses within the legend box.

- *CWE-775, i.e., missing release of file descriptor or handle after effective lifetime, with 672 (5.2 percent) instances:*A file handler is not explicitly closed after it is used, and an attacker can prevent other processes from accessing the resource.
- *CWE-562, i.e., return of stack variable address, with 612 (4.7 percent) instances:*A function call returns an address on the stack, and the value referenced by the address can change unexpectedly.
- *CWE-119, i.e., improper restriction of operations within the Bounds of a Memory Buffer, with 518 (4 percent) instances:*A program can read from or write to a memory location outside of a buffer, and arbitrary code may be executed by redirecting a function pointer to malicious code.
- *CWE-758, i.e., reliance on undefined, unspecified, or implementation-defined behavior, with 482 (3.7 percent) instances:*A property may change its behavior when the software is ported to a different platform.

Certain CWE types, e.g., CWE-119/416/190/476/415, are associated with more security risks in real-world software systems. More specifically, CWE-119/416/190/476/415 have a larger number of related CVE instances, with 12,285/1,073/996/803/151 CVE instances, respectively. Fig. 4 shows the number of the reported CVE instances[23] that are related to each CWE type. A CWE type with a larger number of CVE instances indicates that this CWE type has a more practical impact in terms of the security vulnerability on real-world software systems. For example, 12,328 CVE instances[24] related to CWE-119 – improper restriction of operations within the bounds of a memory buffer, were reported, e.g., by companies who found CWE instances in their code, while only one CVE instance[25] related to CWE-775 – missing release of file descriptor or handle after effective lifetime, was reported. Such $Code_w$ with CWE types that are labeled in red in Fig. 4 probably should be tagged with potential security risks since such CWE types have higher potential impact on real-world software systems.

In addition, we calculate the viewcount of question threads that are associated with CWE-119/416/190/476/415, and find that it is higher than the viewcount for question threads associated with other CWE types – see Appendix B, available in the online supplemental material. Therefore, the former case attracts significantly more traffic, and are more likely to be used by developers. Neglecting the security weaknesses that are associated with CWE types with a large number of CVE instances can expose crowdsourced code snippets with weaknesses and even potentially lead to harmful situations.

Although CWE-908/401 are detected frequently (i.e., 54.2/14 percent instances, respectively), no CVE instance is ever reported for these CWE types. In total, there are 20 CWE types with no reported CVE instance. One possible explanation is that these CWE types are either non-critical in real-world software systems or are easy to detect using security analysis tools during in-house testing, in turn minimizing their security risks in real-world software systems. It is also possible that the CVE database only documents certain types of vulnerabilities, while it does not cover vulnerabilities related to such CWE types. We observe that CWE-908 is the most frequently-detected CWE type in our studied code snippets. In a world of code searching and sharing activities on Stack Overflow, developers can post/share a code snippet in which the variables are not properly initialized, making themselves exposed to potential vulnerabilities. For instance, the uninitialized resource, such as a variable, may contain random values or content that are not properly cleared, which may alter the expected program behavior. Therefore, we suggest that developers pay special attention to the missing initialization when reusing C/C++ code snippets from Stack Overflow.

We observe that of the top 10 CWE types on Stack Overflow in terms of the CWE instance count, CWE-119 and CWE-476 are also in the 2019 CWE top 25 list, with a rank of 1 and 14, respectively. In addition, the top 5 CWE types on Stack Overflow, i.e., CWE-119/416/190/476/772, which are associated with the largest CVE instance count, are all in the 2019 CWE top 25 list, with a rank of 1, 7, 8, 14, and 21, respectively. This finding suggests that the abovementioned CWE types that are prevalent in terms of either CWE count or CVE count have a large security impact.

Furthermore, we observe that CWE-119 that is associated with more than 10K CVE instances contains high severity vulnerabilities with a median CVSS score in the 7.0 – 8.9 range.[26] The CWE types, i.e., CWE-416/190/476/772/415/369, that are associated with 111 – 1256 CVE instances contain medium severity vulnerabilities with a median CVSS score in the 4.0 – 6.9 range. Surprisingly, the CWE types that are associated with fewer than 100 CVE instances contain high severity vulnerabilities, except for CWE-665 that has a median severity score of 5.8.

Overall, the proportion of $Code_w$ doubled from 2008 to 2018. Furthermore, $Code_w$ in certain CWE types, e.g.,

23. The data was obtained on June 28, 2019.

24. https://www.cvedetails.com/vulnerability-list/cweid-119/vulnerabilities.html

25. https://www.cvedetails.com/vulnerability-list/cweid-775/vulnerabilities.html

26. https://nvd.nist.gov/vuln-metrics/cvss

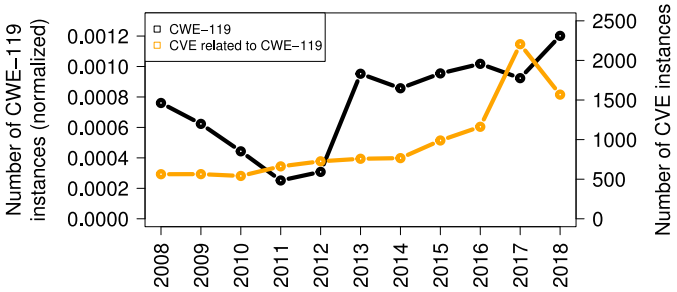

Fig. 5. The growth of the number of CWE-119 instances normalized by the number of C/C++ code snippets in each year. The growth of CVE instances related to CWE-119 is also shown.

CWE-119, has increased in recent years. The growth trend of $Code_w$ and CWE instance count in each individual CWE type are shown in Appendix C, available in the online supplemental material. To further understand the impact of such growing CWE types, we analyze the trend of CWE-119, which is the CWE type with the largest number of CVE instances in real-world software systems, and its related CVE instances.[27] We observe that the number of CWE-119 instances dropped from 2008 to 2011 and since then had a rising trend, although in both 2014 and 2017 the number of CWE instances dropped from the preceding years, respectively. The corresponding CVE instances were increasing until 2017 as shown in Fig. 5. In particular, the severity level of CWE-119 is considerably higher than other CWE types. Therefore, we suggest that developers be cautious about these CWE types when reusing code snippets from Stack Overflow answers, especially paying attention to the boundary of memory buffers when reusing C/C++ code snippets on Stack Overflow.

We observe that the median score of answers with different numbers of CWE instances is either zero or one. There is no significant correlation between the answer scores and the number of associated CWE instances (p-value = 0.01). Thus, there is no difference in the scores of answers that contain more or less weakness. In addition, out of the 12,998 CWE instances in the latest version of code snippets with weaknesses, 9,535 (i.e., 73.4 percent) are in non-accepted answers, showing that the majority of the code weaknesses are in non-accepted answers.

We identify 36 percent (i.e., 32 out of 89) CWE types in Stack Overflow answers. CWE-908 – *use of uninitialized resource*, accounts for 54.2 percent of all the CWE instances. In particular, some types of the detected CWEs, such as CWE-119 – *improper restriction of operations within the bounds of a memory buffer*, are common in code snippets and are common weaknesses in real-world software systems. We also observe that the proportion of $Code_w$ doubled from 2008 to 2018, and that the number of instances in some CWE types, e.g., CWE-775/119/685, is rising in recent years.

TABLE 2
The Number/Proportion of $Code_w$ versus Different Number of Code Revisions

#Revisions	#$Code_w$	#Unchanged	#Improved	#Deteriorated
0	8,103	NA	NA	NA
≥ 1	3,645	1,886 (51.7%)	1,218 (33.4%)	541 (14.8%)
1	2,369	1,340 (56.6%)	714 (30.1%)	315 (13.3%)
2	774	349 (45.1%)	294 (38.0%)	131 (16.9%)
≥ 3	502	197 (39.2%)	210 (41.8%)	95 (18.9%)

4.2 RQ2: How Does Code With Weaknesses Evolve Through Revisions?

92.6 percent (i.e., 10,884) of the 11,748 $Code_w$ has weaknesses introduced when their code snippets were initially created on Stack Overflow, and 69 percent (i.e., 8,103 out of 11,748) of the $Code_w$ has never been revised, as shown in Table 2. For example, an answer[28] recommended the use of the *strcpy* function to solve an error when checking if words in an array of pointers to char are the same as words in a function, although a comment pointed out that "*this is bad, it uses scanf and strcpy unsafely, which causes buffer overflows, a very serious security vulnerability*", the answer was never revised.

In general, more rounds of code revisions are more likely to reduce code weaknesses. In 31 percent (i.e., 3,645) of $Code_w$, there are 6,831 $Version_w$, that is, code versions with weaknesses. In these 3,645 code snippets with weaknesses, there are a total number of 9,488 code versions. Table 2 lists the number of $Code_w$ with different revision numbers. Note that a code snippet with zero revision has one version in its history. The proportion of *improved* $Code_w$ increases as the number of revisions increases, indicating that *having more code revisions is beneficial in reducing security weaknesses*. More specifically, as the number of revisions increases from one to $\geq$ three, the proportion of *improved* $Code_w$ increases from 30.1 to 41.8 percent. We also perform a Mann-Whitney test for both the *improved* and *deteriorated* $Code_w$ with different revision numbers, and find that for each case the last code version is significantly different from the first code version in terms of the number of CWE instances for each $Code_w$ (p-value $<$ 0.05). To illustrate how users revised their code snippets leading to a reduce of weaknesses, we observe an accepted answer[29] that was revised five times. The initial code snippet in the answer contained CWE-562 – *return of stack variable address*, while the latest code snippet no longer contained any weakness. The answerer also posted two comments under the question to explain that "*I had to fix my answer so dest wasn't statically allocated ...*", and to further warn that "*... to reduce risk of misuse and memory leaks. A common problem is that people may come back later to make changes 'borrow' a function for quick use in another feature of the program or another solution and miss that fact and introduce a leak. So it's a good precautionary action.*"

In $Code_w$ with different rounds of code revisions, a larger proportion of code snippets have reduced rather than increased the number of their associated security weaknesses. To illustrate, an answer[30] created a file but never closed it. Later, another

27. The data is collected by crawling the CVE information website on October 9, 2019 at https://www.cvedetails.com/vulnerability-list/cweid-119/vulnerabilities.html

28. https://stackoverflow.com/a/52807726/
29. https://stackoverflow.com/a/41971294/
30. https://stackoverflow.com/revisions/18637122/3/

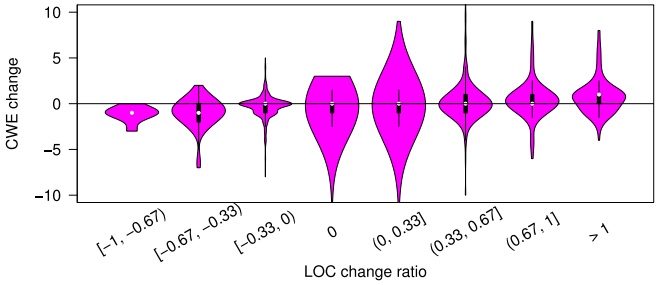

Fig. 6. The distribution of the change of CWE instances number across different ranges of LOC change ratio. The negative CWE change occurs within the *improved* $Code_w$. The zero CWE change occurs within the *unchanged* $Code_w$. The positive CWE change occurs within the *deteriorated* $Code_w$.

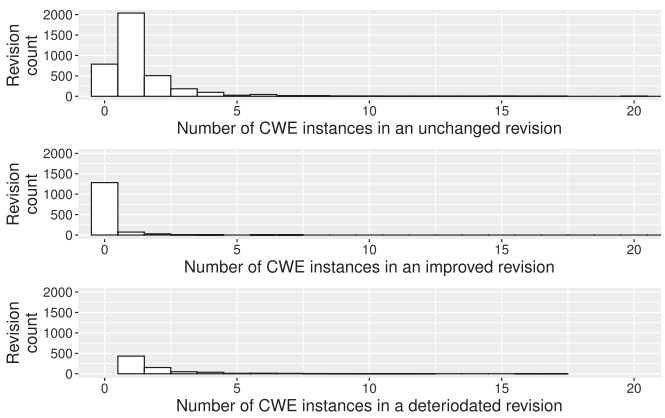

Fig. 7. The distribution of code revisions with different number of CWE instances in an unchanged, improved, and deteriodated revision, respectively.

user edited the code to properly close the file in order to release the file handler. Table 2 shows that the *improved* $Code_w$ are at least twice more than the *deteriorated* $Code_w$ for different numbers of code revisions. In the circumstances when code revisions introduce more weaknesses, i.e., *deteriorated* $Code_w$, the proportion of *deteriorated* $Code_w$ slightly increases from 13.3 to 18.9 percent as the number of code revisions increases from one to $\geq$ three. Compared with the *improved* $Code_w$, the *deteriorated* $Code_w$ have more CWE instances that are possibly related to the addition of more LOC to existing code snippets. To test this assumption, we investigate the relationship between the change of lines of code and the change of number of CWE instances from the first to the last version of a code snippet. We calculate the LOC change ratio using the following equation:

$$LOC\ change\ ratio = \frac{LOC(last) - LOC(first)}{LOC(first)},$$

where $LOC(first)$ and $LOC(last)$ present the LOC in the first and last versions, respectively. We calculate the change of the CWE instance number as the change of CWE instances from the first code version to the last code version

$$CWE\ count(last) - CWE\ count(first).$$

We find that *the higher the proportion of LOC change ratio, the more the number of CWE instances changes, suggesting that there is a positive correlation between the LOC change ratio and the CWE change.* The distribution of the change of CWE instances number in $Code_w$ with different ranges of LOC change ratio is shown in Fig. 6. Code weaknesses are more likely to be detected and a Stack Overflow user is exposed to a higher security risk, especially after revising an answer by adding more code. We cannot make any data-supported conclusion of the reasons for this observation, but one possible explanation is that the user is not aware of the security consequences of the provided content, and the chance of having risk increases when more code is given.

We compare $Code_w$ and code snippets without weaknesses in terms of their revision number to exam whether $Code_w$ are more likely to be revised. Among all the 646,716 code snippets that are scanned by Cppcheck, 20.1 percent (i.e., 130,100) have been revised, while a larger proportion, that is, 31 percent (i.e., 3,645 out of 11,748) of $Code_w$ have been revised, indicating that $Code_w$ *are more likely (54.2 percent) to be revised*. The rest 69 percent of the $Code_w$ have never been revised. Weaknesses are present in these code snippets as they are created and no further action is ever performed. Khandelwal *et al.* conducted a survey to understand whether gamification helps in peer code review activity, and found that 54 percent of the correspondents were in strong favour of gamification in code review [33]. Therefore, future research may study whether a gamification mechanism can encourage code review and thus alleviate the risk of code weaknesses on Stack Overflow.

We calculate the change of CWE instances in two consecutive versions in the 3,645 code snippets with at least two code versions (i.e., 5,843 code revisions in total). In the 5,843 code revisions, 3,728 (i.e., 63.8 percent) of them have the same number of CWE instances compared with the preceding code version. 1,400 (i.e., 24.0 percent) of the revisions have fewer CWE instances compared with the preceding code version. 715 (i.e., 12.2 percent) of the revisions have more CWE instances compared with the preceding code version. Therefore, the majority of the consecutive code revisions (i.e., 63.8 percent) do not change code weakness. In the rest (i.e., 36.2 percent) of the code revisions, revisions are more likely to decrease the code weakness than to increase the code weakness. Among all the consecutive code revisions, 24 percent correct code weaknesses, while after multiple code revisions, i.e., comparing the last version with the first version, 33.4 percent eventually correct code weaknesses. Therefore, more weaknesses are eventually fixed even though they were not fixed in earlier revisions. Fig. 7 shows the distributions of code revisions with different numbers of CWE instances. We observe that the time between the first and last version with weaknesses is much shorter than the time from the last version with weaknesses to present time,[31] as shown in Appendix D, available in the online supplemental material. In other words, users do not see much of the earlier versions compared to the latest version, and the latest version is the final version that is presented to users.

We further examine the impact of code revisions on the number of CWE instances within each CWE type. For each specific CWE type, we extract the number of CWE instances in the last code version, i.e., C_{last}, the number of CWE

31. As of December 2018, when our data was collected.

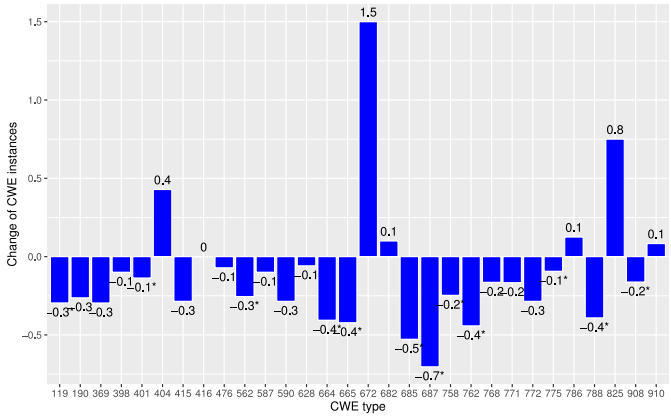

Fig. 8. The change of CWE instances for different CWE types. * indicates that the change is statistically significant (i.e., p-value < 0.05).

instances in the first code version, i.e., C_{first}, and calculate the change of CWE instances as: $(C_{last} - C_{first})/C_{first}$.

For the majority of CWE types, revisions of $Code_w$ reduce the number of CWE instances. For each CWE type, we extract all the $Code_w$ that contain such CWE instances, and compare the number of CWE instances in this CWE type between the first and last version of the same $Code_w$. As shown in Fig. 8, we observe that the CWE instances increase for only 6 CWE types, i.e., 404/672/682/786/825/910, while it drops for 24 CWE types. To test if the difference is statistically significant, we perform a Wilcoxon signed-rank test for each CWE type by comparing the number of CWE instances between the first and last version (i.e., paired comparisons). In Fig. 8, we label the change of CWE instances with a $*$ sign when the difference is statistically significant (i.e., p-value < 0.05). Note that none of the increases in the number of CWE instances for CWE-404/672/682/786/825/910 is statistically significant, indicating that the revisions of $Code_w$ do not increase the number of CWE instances significantly. On the other hand, the *improved* $Code_w$ have a statistically significant drop in their number of CWE-119/401/562/664/665/685/687/758/762/775/788/908, indicating the revision of such $Code_w$ leads to a decrease in the security weaknesses. Note that for the rest of CWE types with a drop in their last code versions, the revision does not lead to a decrease in the security weaknesses, possibly due to the small number of CWE instances in such cases, i.e., a median value of 20, and a maximum value of 78. We encourage users to pay extra attention to the rest of these CWE types.

In summary, we observe that code revisions are associated with a reduction in the number of security weaknesses in code snippets in general; however, some code revisions can also introduce weaknesses. We suggest online weakness detection tools to be used to identify $Code_w$. We find that the majority of $Code_w$ remain unchanged to the community: they are never revised and may be reused by others. We also observe that 4,271 (i.e., 36.4 percent) of $Code_w$ are in an answer revision, i.e., weaknesses still exist even after an answer is revised. We suggest Stack Overflow to use better incentives together with weakness detection tools to motivate users to actively reduce code weaknesses through answer revisions, in turn improving the crowdsourced code quality from a security perspective.

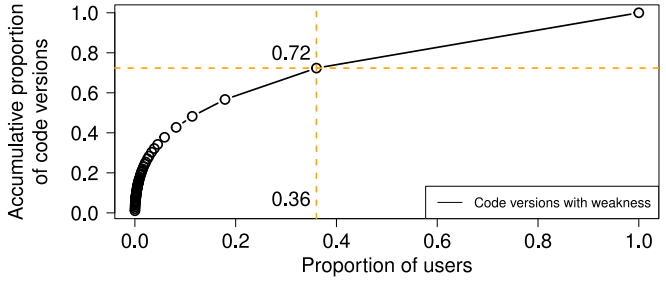

Fig. 9. The accumulative proportion of $Version_w$ that were posted by the proportion of users.

We observe that $Code_w$ without any revisions are less likely to have comments in the associated answer, while $Code_w$ that are eventually improved have more comments in the associated answer – see Appendix E, available in the online supplemental material. This finding suggests *the positive effect of discussions through commenting on improving the quality of code snippets*. As shown in Fig. 3, a comment pointed out that the *strlcpy/strlcat* functions are insecure in an answer although the answer was never edited. Users are recommended to read through the associated comments in an answer in case security knowledge is added by the community through commenting.

More rounds of code revisions are associated with a reduction in the number of code weaknesses. A larger proportion of $Code_w$ have reduced rather than increased the number of security weaknesses. There is a positive correlation between the LOC change ratio and the CWE change number. In the majority of CWE types, code revisions reduce their associated CWE instances. Especially for CWE-119/401/562/664/665/685/687/758/762/775/788/908, the number of weaknesses drops significantly. In addition, $Code_w$ are more likely (54.2 percent) to be revised than code in general. However, the majority of $Code_w$ have weaknesses introduced in the first code version and they are never revised.

4.3 RQ3: What are the Characteristics of the Users who Contributed to Code With Weaknesses?

The majority of the C/C++ $Version_w$ were contributed by a small number of users. 72.4 percent (i.e., 10,652) of $Version_w$ were posted by 36 percent (i.e., 2,292) of users, as shown in Fig. 9. 64.0 percent (i.e., 4,070) of the users who contribute $Version_w$ have contributed only one $Version_w$. *Among all the 85,165 users who posted C/C++ code snippets, only 7.5 percent (i.e., 6,361) of them posted code snippets that have weaknesses.*

More active users are less likely to introduce $Code_w$. Fig. 10 illustrates the relationship between the number of code revisions contributed by a user and the weakness density of a user's code. The figure shows that *the weakness density of a user's code drops when the number of contributed code revisions by the user increases*. In particular, 15.1 percent (i.e., 958) of the users contribute only one code version ever and it is a $Version_w$, as shown by the top left point in Fig. 10. In Fig. 11, we compare the reputation of the contributor for a code version with different numbers of CWE instances in the code

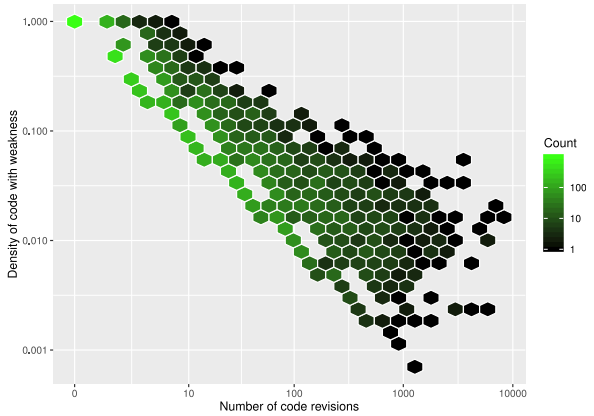

Fig. 10. As the number of code revisions increases for a user, the density of contributed $Version_w$ by that user drops.

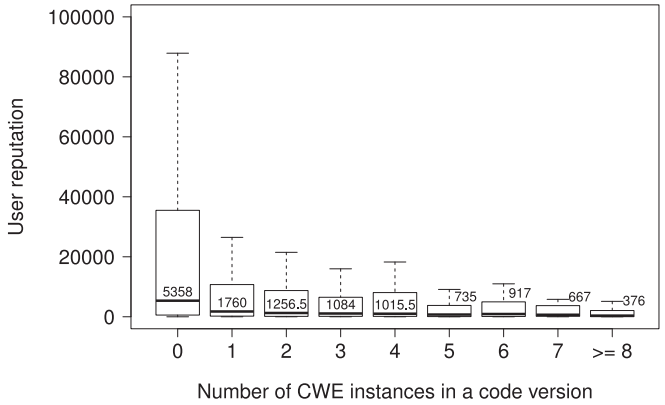

Fig. 11. The distribution of user reputation points for users who contribute code versions both without and with different numbers of CWE instances.

version, which shows that *users with higher reputation tend to introduce fewer CWE instances in their contributed code versions*. As an example, one user has only 6 $Version_w$, while the current reputation of that user is more than 500K.[32] In particular, we show the median user reputation for each group of contributors, and the medium reputation of the contributors to code versions without weaknesses are at least three times higher than the medium reputation of the contributor to $Version_w$. We run a Mann-Whitney test and observe that the difference in contributors' reputation between code versions without weaknesses and $Version_w$ is significant with a p-value < 0.05.

In total, 78.0 percent of users contribute code with only one CWE type. Furthermore, 42.2 percent (i.e., 2,686) of the users contribute only one CWE instance in all their $Version_w$, as shown in Fig. 12. 81.8 percent (i.e., 5,206) of the users contribute less than five CWE instances in all their $Version_w$.

Users Tend to Commit the Same Types of CWE Instances Repeatedly. For each CWE type, we show the distribution of the number of contributed CWE instances by different users in Fig. 13. The figure suggests that certain Stack Overflow users repeatedly contribute code snippets with specific CWE types. For example, we observe that in CWE-401/775/908 some users contribute code with such CWE types for more than 10 times. In other words, users may not even realize that they are posting code snippets with the same potential security weaknesses repeatedly. For example, one user has contributed CWE-775, i.e., *missing release of file descriptor or handle after effective lifetime*, for 79 times.[33] Furthermore, we observe that the timespan when users repeatedly post the same CWE type is short for the majority of the CWE types, i.e., a median value of less than one day except for CWE-562 – see Appendix F, available in the online supplemental material. Future tooling support to identify Stack Overflow code vulnerability can actively support users who repeatedly contribute the same code weakness type.

To better understand how users contribute different CWE types, we analyze the users who actively contribute code weaknesses, i.e., at least five CWE instances, in their $Version_w$. To understand the CWE types that are contributed by each user, we calculate the normalized entropy of the CWE types from his/her posted $Code_w$. More specifically, we wish to measure whether the CWE types that are contributed by a user are concentrated on a small number of CWE. We count the number of CWE instances for each CWE type, and calculate the normalized entropy of the resulting distribution. Fig. 14 shows the distribution of the normalized entropy for 1,153 users who contributed at least 5 CWE instances. An entropy value of 0 indicates that the user only contributed a single type of CWE in their code snippets. We observe that *37.7 percent of users are likely to introduce a single type of CWE instances in their posted code versions*.

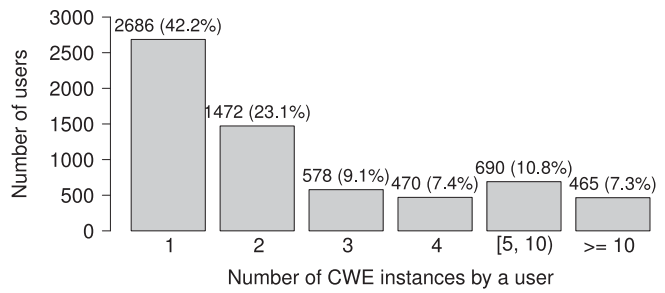

Fig. 12. The number/proportion of users who contribute a different number of CWE instances.

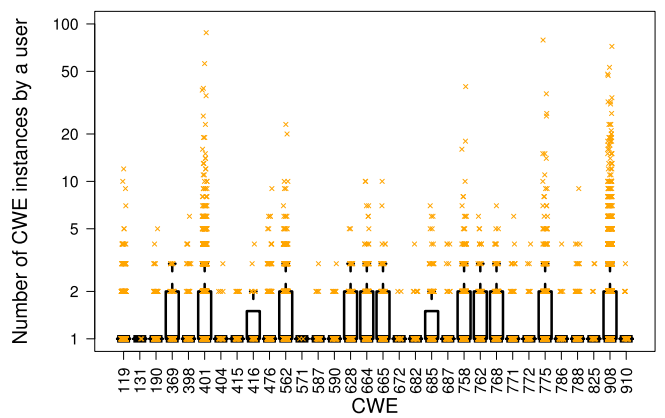

Fig. 13. The distribution of contributed CWE instances by different users for different CWE types.

32. https://stackoverflow.com/users/505088/
33. https://stackoverflow.com/users/3422102/

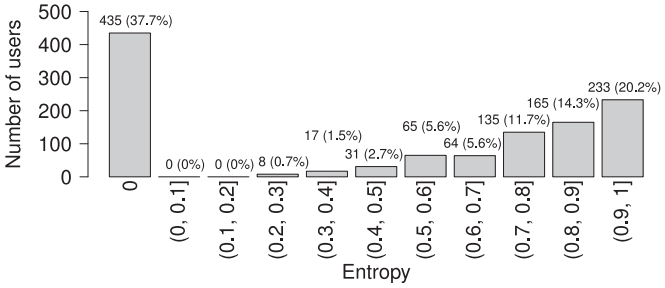

Fig. 14. The distribution of entropy for CWE instances of different types.

The majority (i.e., 72.4 percent) of $Version_w$ were contributed by 36 percent of the users who contributed $Version_w$. Only 7.5 percent of users who contributed C/C++ code snippets had code weaknesses. Users contribute fewer $Version_w$ as their activities increase in terms of both the number of contributed code versions and the gained reputation. 78.0 percent of the users contribute only one CWE type, and users tend to have the same CWE type repeatedly in their contributed code.

5 Implications of Our Findings

5.1 How Does the Stack Overflow Community Respond to Security Issues of C/C++ Code Snippets?

In our qualitative study to understand how the Stack Overflow community responds to code weaknesses, we observe that 25 out of our 40 (i.e., 62.5 percent) suggested revisions are adopted by the Stack Overflow community, i.e., approved by either the answerers themselves or the Stack Overflow moderators. For example, we fixed a potential memory leak caused by inappropriate use of realloc in an answer,[34] which has been adopted by the answerer. Note that no reason was provided when a suggested revision was approved.

For the suggested revisions that were rejected, the reasons for the rejections can be: "*this edit was intended to address the author of the post and makes no sense as an edit, it should have been written as a comment or an answer*." For example, our attempt[35] to fix a potential resource leak was rejected by Stack Overflow moderators based on the above-mentioned reason. Note that most of the rejected revisions were done by Stack Overflow moderators. Although they considered the revisions should have been contributed in a different format, they agreed that the revisions addressed security weaknesses. From our experiment, we do note that the posting of a comment before suggesting the actual code revisions can improve the chance of the suggested revisions being adopted [28]. Our experiment indicates that the effort of fixing the detected code weaknesses are acknowledged by the Stack Overflow community; however, we observe that even for the same code weakness, one suggested revision in an associated answer can be approved while a suggested revision in another answer can be rejected. Some moderators may not be aware of the risk of weaknesses in code snippets, and they are possibly rejecting code revisions that aim to improve the security of the shared code snippets on Stack Overflow. In the 15 out of our 40 suggested revisions that were rejected, the different reasons are summarized in Appendix G, available in the online supplemental material.

The feedback from the Stack Overflow community highlights the need of a tool to automatically detect and correct $Code_w$, which can identify vulnerable C++ code snippets and warn the user with both explanation and mitigation of the vulnerability as proposed in [6].

5.2 Implications for Stack Overflow Users

Table 3 shows our major findings from empirically mining the C/C++ $Code_w$ on Stack Overflow and their implications.

We find that certain types of weaknesses, e.g., CWE-119, are common on Stack Overflow while also popular in real-world software systems. We suggest users to pay attention to operations at the bounds of the memory buffer in their code.

Overall, users should be cautious when reusing code from Stack Overflow, since we find the proportion of $Code_w$ have doubled during 10 years, i.e., 2008 – 2018. Furthermore, the frequency of CWE-775/119/685 is rising in recent years. Certain users are repeatedly contributing the same type of weaknesses. New mechanism can be designed to alert such users about their security issues when they contribute Stack Overflow code snippets. Based on our finding that code revisions are associated with a reduction in code weaknesses, additional efforts are needed to better assist developers with removing code weaknesses.

We observe that more active users on Stack Overflow contribute fewer $Code_w$. Code revisions, especially the ones that reduce code weaknesses, can be rewarded by extra reputation points and/or badges to improve the quality of the crowdsourced code snippets on Stack Overflow. Note that prior work [5] observe that the user reputation does not correlate with insecure Python code snippets on Stack Overflow. One assumption is that different programming languages may have different phenomenons. We encourage future research to investigate the usage patterns of code vulnerability across different programming languages.

Furthermore, the answerers who contributed more code weaknesses are less active since we observe that more active users contributed fewer $Version_w$. We suggest that Stack Overflow can scan the code snippets online when users post answers. Therefore, inactive users can be alerted about their potentially insecure code before they contribute such code in their answers. Our qualitative study of 40 $Code_w$ demonstrates the value of online code review to actively improve the security quality of the code base on Stack Overflow, together with commenting on code weaknesses.

Currently, it is up to the users themselves to decide whether code snippets are secure enough to share and/or reuse on Stack Overflow. To illustrate the current situation of code security on Stack Overflow, we show a Stack Overflow META discussion.[36] In the question, the asker cited that "*Internet resources such as Stack Overflow are blamed for*

34. https://stackoverflow.com/review/suggested-edits/24247281
35. https://stackoverflow.com/review/suggested-edits/24246951
36. https://meta.stackoverflow.com/q/356892/

TABLE 3
Our Major Findings From Empirically Mining the C/C++ $Code_w$ on Stack Overflow and Their Implications

Code weakness types – Section 4.1	**Implications**
Cppcheck, which is able to detect 59 out of all the 89 C/C++ CWE types, reports 32 C/C++ CWE types in Stack Overflow code snippets. 12,998 CWE instances are detected in the latest versions of 7,481 answers.	Code weaknesses are detected in 2% of the C/C++ answers with code snippets. Stack Overflow can perform CWE scanning for all the code that is hosted on its platform.
CWE-119/416/190/476/415 are associated with security risks in real-world software systems. The viewcount of question threads with these CWE types are higher than question threads that are associated with other CWE types.	Certain CWE types have higher risks, thus users should pay special attention to such security issues.
The proportion of $Code_w$ grows year by year and doubles from 2008 to 2018.	The security issue on Stack Overflow should be alerted since it is growing over time.
Evolution of $Code_w$ through code revisions – Section 4.2	**Implications**
92.6% of the 11,748 $Code_w$ have weaknesses introduced when initially created.	Developers should pay attention to the security aspects of their code when they post answers.
More code revisions are associated with a reduction of code weaknesses. In the majority of CWE types, code revisions are associated with a reduction in the number of CWE instances.	Stack Overflow should encourage better code review mechanisms and motivate users to revise code with possible security vulnerabilities.
31% of $Code_w$ were revised, while 20.1% of C/C++ code snippets were revised.	Users make revisions to $Code_w$ 54.2% more than code in general.
$Code_w$ contributor characteristics – Section 4.3	**Implications**
7.5% of the 85,165 users posted C/C++ code snippets have contributed code weaknesses.	A small group of users contribute code weakness.
More active users contribute fewer weaknesses.	The Stack Overflow code snippets can leverage user activity level to prioritize code reviewing activities.
Users tend to commit the same CWE types repeatedly.	Future tooling is needed to help users improve their code by alerting them about weaknesses in their code contributions.

promoting insecure solutions that are naively copypasted by inexperienced developers", and asked that "*does this mean we should do something about it or is it all the developers fault?*". In the two associated answers within this question, one answer starts with "*no, we don't have to change a thing*", while the other starts with "*I don't think we have to change anything systematically*." Thus, on Stack Overflow META, the opinions of how code security should be maintained show that both the community and the platform do not need to be responsible for improving code security. Our study analyzes how the improvement of code security is done in real-world practices on Stack Overflow. We observe more than 10,000 code weaknesses in our experiment. Our qualitative study demonstrates that actively tagging code with weaknesses is accepted by the community, leading to an improvement of the crowdsourced code quality.

6 Threats to Validity

External Validity. In this study, we focus on C/C++ code snippets, while code snippets in other programming languages may have distinct characteristics from our findings of C/C++ code snippets. We encourage future research to investigate security weaknesses in other programming languages.

Furthermore, we only investigate code snippets from Stack Overflow answers. Note that code snippets from questions can also have weaknesses. Since questions are posted by askers to seek solutions for their problems, the code snippets in questions are probably problematic to start with. However, code snippets in answers are posted by users who aim to solve issues. Their code snippets are shared more frequently. Therefore, we study code snippets in answers to provide security related insights.

To mitigate the bias from pseudo code or command line functions, we remove code snippets with less than five lines of code. We may lose a portion of code snippets. However, we do note that there is no standard way to determine the threshold for removing such code snippets, and we follow prior studies [16], [17] in using a threshold of five lines of code. In addition, we use the Guesslang tool to determine whether a code snippet is written in C/C++. Guesslang is based on a deep learning model trained with source code files. Although the accuracy of the tool is evaluated to be 91 percent from our 100 randomly sampled code snippets, around 10 percent of our collected C/C++ code snippets can be false positives. Code snippets that are not in C/C++ can be introduced in our study and may bias our understanding of code weaknesses in Stack Overflow answers. We encourage future research to develop more accurate techniques to identify C/C++ code snippets.

We detect code snippets with weaknesses using Cppcheck. Cppcheck can identify 59 out of the 89 types of C/C++ code weaknesses. The results that are generated by Cppcheck can contain false positives, which may bias our results, although it aims to minimize false positives.[37] In order to understand the bias of studying the code snippets with Cppcheck, we manually evaluated the accuracy of Cppcheck in Section 3.1.2 and found that it has an accuracy of 0.85.

Internal Validity. When we scan code snippets with Cppcheck, we skip any code snippet that returns syntax errors. Such code snippets are probably code segments that miss a substantial part of the compilable code, thus are not included in our study. This approach may not capture all types of security weaknesses in C/C++ code snippets on Stack Overflow. We evaluate users' activity level by their reputation points on Stack Overflow. It is possible that a user has gained reputation points by activities in other tags. Thus, the gained reputation points may not accurately

37. http://cppcheck.sourceforge.net/manual.pdf

reflect the activity level in C/C++ programming languages. To complement this, we also consider the number of code revisions that a user has done as the proxy of the activity level of the user.

We note that there is no single approach that can fully specify the impact of code weaknesses. To provide a thorough view of the impact of code weaknesses, in this study we evaluate the impact of different CWE types by referring to both the CVSS scores and the 2019 CWE top 25 list to provide an overview picture of the impact of CWEs in real world projects. While the CVSS score assesses the severity of a CVE instance, the 2019 CWE top 25 list characterizes the impact of a code weakness that can potentially lead to software vulnerabilities. The CWE top 25 list is provided by the CWE team in evaluating the impact of each CWE type. This approach uses vulnerabilities that have CVE records in the *National Vulnerability Database* – NVD. However, even in the same top 25 list, some CWE types may have a higher severity score than others, and weaknesses that are not included in the list can still have an impact while being underrepresented in the CWE top 25 list. In addition, the vulnerabilities that are reported by their vendors in the CVE dataset may not represent all the security risks that developers encounter. Last but not least, we use the CWE top 25 list that was published in 2019, while the list itself can evolve over time with their rankings changed or certain CWE types added/removed in the future. Although the majority (i.e., 22) of the CWE types in 2019 still remain as the top 25 in 2020, the interpretation of our analysis reflects the exposure of code weaknesses in 2019. We encourage future work to evaluate the impact of code weaknesses on Stack Overflow, for example, by considering an industry standard third party independent list of security impact, or conducting user surveys. Future work can also study the evolution of code weaknesses over time.

Construct Validity. Threats related to the construct validity is related to how we define a code snippet as unchanged/improved/deteriorated in terms of its weakness. In our study, we use the number of CWE instances that are detected by Cppcheck as a proxy to measure the quality of a code snippet. However, certain CWE types are more severe than other CWE types. Thus, a quantitative measure, e.g., CWE count, that indicates the quality of code may be biased, as one CWE instance may be more severe than multiple CWE instances combined. To evaluate the impact of code weaknesses, we also show in Fig. 4 the median CVSS score, i.e., a score to represent the severity of software vulnerabilities, of CVE instances that are crawled from cvedetails.com in July 2020 in each CWE type in Section 4.1. According to NVD,[38] a CVSS score of 4.0 to 6.9 is considered medium severity, and a CVSS score of 7.0 to 10.0 is considered high severity. To further understand how code weaknesses are evolving, we also analyze the change of CWE instances for different CWE types in Fig. 8. Furthermore, in our experiment of reporting the results of our scanned code weaknesses on Stack Overflow, 62.5 percent of the identified weaknesses were acknowledged and addressed by users, indicating that Cppcheck can detect code weaknesses that are of concern by the Stack Overflow community.

Another threat to our construct validity is about how we measure the activity level of a user. It is challenging to measure a user's activities on Stack Overflow. In our study, we use both code revision count and reputation points as proxies to measure user activities. Although these two proxies may introduce bias, our results that are observed based on them are aligned. Future research can explore other metrics to characterize user activities and understand how different users post insecure code snippets on Stack Overflow.

In RQ3, we select the users that have contributed more than five CWE instances. The threshold selection could be a construction threat to our results. To mitigate the threat, we set the threshold of at least two CWE instances from a user and observed that our result still holds. For 3,666 users who contributed at least two CWE instances, the majority (i.e., 61.8 percent) of users are likely to introduce a single CWE type in their posted code versions.

7 Related Work

7.1 Security in Software Systems

Code security is a critical issue in software engineering. Vulnerable code can undermine the quality of software systems. A remarkable research effort has been invested in the security issues of software systems. For example, Pletea *et al.* found that 10 percent of discussions on GitHub are related to security [34]. Acar *et al.* surveyed security guidance resources on the web to inform developers about how to write secure code [35]. Especially, C/C++ security issues are commonly studied in the literature [36], [37], [38], [39], [40], [41], [42], [43]. Alnaeli *et al.* analyzed how vulnerable source code is used in 15 C/C++ software systems. They showed that vulnerable functions, such as *strcmp*, *strlen* and *memcpy*, play a major roles in unsafe code [42]. Mcheick *et al.* proposed a tool to detect memory management and type errors in C/C++ based on runtime information [40]. Yang *et al.* proposed an approach in a commercial security analysis tool to assist developers in fixing software vulnerabilities [44]. Different from prior studies that focus on security at the system level, we focus our study on code snippets.

7.2 Studying Code on Stack Overflow

For more than 10 years, Stack Overflow has accumulated questions and answers related to programming, including millions of code snippets. Stack Overflow code snippets are valuable resources for developers. They have been actively studied in the software engineering community. Yang *et al.* studied the usability of code snippets across C#, Java, JavaScript and Python [15]. They observed that Python and JavaScript code snippets are more parsable/runnable than Java/C#. An *et al.* analyzed 399 Android apps to investigate potential license violations when developers reuse code snippets from Stack Overflow and from Android apps into Stack Overflow [13]. Treude *et al.* surveyed how Stack Overflow code snippets are self-explanatory [45]. Campos *et al.* analyzed JavaScript code snippets on Stack Overflow and flagged violations, such as errors and stylistic issues [46].

To better understand whether security issues exist and how they are present in the crowdsourced knowledge of

38. https://nvd.nist.gov/vuln-metrics/cvss

Stack Overflow, prior studies investigated various security aspects on Stack Overflow. For example, Yang *et al.* investigated security-related questions on Stack Overflow. They identified both popular and difficult topics related to security that are asked on Stack Overflow [47]. Barua *et al.* explored the discussion topics on Stack Overflow and identified security as a diverse topic that crosses multiple domains [48]. Fischer *et al.* analyzed insecure code snippets related to Android on Stack Overflow, and found that 15 percent of 1.3 million Android applications contained insecure code snippets from Stack Overflow [2]. Meng *et al.* inspected Stack Overflow threads that are related to Java security and identified the root causes and solutions for Java secure coding [4]. Chen *et al.* extracted Stack Overflow code snippets related to Java security and observed that at least 41 percent of their inspected security-related answers are insecure [31]. Acar *et al.* surveyed the security quality of Stack Overflow threads and observed that although Stack Overflow crowdsourced knowledge is accessible compared with official API documentation it often leads to insecurity [49]. Lopez *et al.* conducted a study to examine how users ask questions related to security on Stack Overflow. They found that security conversations are rich, and some askers and commenters are actively involved in such conversations [50]. Rahman *et al.* analyzed Python code snippets on Stack Overflow and found that they suffered from insecure coding practices such as code injection [5].

Prior studies examined vulnerabilities on Stack Overflow for Android [2], Java [4], and Python [5]. Different from these studies, our study focuses on C/C++ code snippets, which is the programming language with the most CWE types out of all programming languages – giving us a much larger number of observations. We have a number of observations. For instance, a prior study [5] notes that the user reputation does not correlate with insecure Python code snippets on Stack Overflow, which is the opposite of our finding. In their study, user reputation is normalized by the membership period of a user. We also calculated the normalized user reputation and our findings still hold in terms of the reputation points normalized by time – more active users contribute fewer C/C++ code weaknesses. Different programming language communities may have different user contributions in terms of code weaknesses. In addition, we studied how code weaknesses evolve through revisions and the characteristics of the contributors of these code weaknesses.

C/C++ code weaknesses on Stack Overflow have not been extensively studied in prior work. In [6], Verdi *et al.* observed that the number of both CWE and vulnerable answers drops over the years.[39] Different from their study, we observe that the proportion of code snippets with C/C++ weaknesses are increasing over the years from 2009 to 2018. The difference in the research design may contribute to the differences between the two findings. For example, Verdi *et al.* studied answers with the C++ tag, while in our study answers from both the C and C++ tags are examined. In our qualitative study, we observe that users do care about code weaknesses, i.e., Stack Overflow moderators/users frequently fixed the code weaknesses that we reported. Our study highlights the need for a better mechanism to improve the management of $Code_w$, especially given that the number of code weaknesses is increasing over time but weakness maintenance efforts are not increasing proportionally, leading to Stack Overflow becoming a more and more insecure platform for crowdsourced knowledge sharing.

In addition, Verdi *et al.* analyzed the prevalence of the migration of vulnerable C++ code snippets from Stack Overflow to GitHub [6], while our study focuses on all C/C++ code snippets on Stack Overflow, including both code snippets that are possibly migrated to GitHub and code snippets that are not migrated to GitHub. Code snippets with weaknesses may pose a risk even if they have yet to migrate to GitHub. For example, code snippets may be already in use by commercial systems or other open source systems that are not hosted publicly on GitHub; in addition, such vulnerable code may get migrated to GitHub in the future. Besides analyzing the vulnerability types, we also analyze the revision history of $Code_w$ and users that contribute such code snippets. We obtain insightful observations, e.g., revisions do help reduce the number of CWE instances, and some contributors repeat the same CWE throughout their different contributions.

In this paper, we conduct a large-scale empirical study to analyze all code snippets in Stack Overflow answers tagged with C/C++. We are the first study that studies the weaknesses of C/C++ code, which is known to have the most security vulnerabilities [10]. We provide insights to developers so their code sharing activities lead to fewer security risks.

8 Conclusion

Code snippets on Stack Overflow are shared widely by developers and code security is a critical condition for reuse. In this study, we investigate the weakness of C/C++ code snippets on Stack Overflow by scanning 646,716 C/C++ code snippets in Stack Overflow answers using Cppcheck. We identified 32 types of code weaknesses on Stack Overflow, and observed that some CWE types, i.e., CWE-119/416/190/476/415, are also associated with many vulnerabilities in real-world software systems. In order to explore how $Code_w$ evolve, we analyze the revision history of such $Code_w$ and find that more code revisions are associated with a reduction of code weaknesses. Our analysis also shows that more active users contribute fewer $Version_w$. Our findings can be leveraged by future studies to improve the quality of Stack Overflow's crowdsourced code snippets.

References

[1] Y. Wu, S. Wang, C.-P. Bezemer, and K. Inoue, "How do developers utilize source code from stack overflow?," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 637–673, 2019.

[2] F. Fischer *et al.*, "Stack overflow considered harmful? The impact of copy & paste on Android application security," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 121–136.

[3] Wikipedia, "Vulnerability (computing)." Accessed: Feb. 01, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Vulnerability_(computing)

[4] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty, "Secure coding practices in Java: Challenges and vulnerabilities," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 372–383.

[5] A. Rahman, E. Farhana, and N. Imtiaz, "Snakes in paradise?: Insecure python-related coding practices in stack overflow," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 200–204.

39. Fig. 6 from https://arxiv.org/abs/1910.01321

[6] M. Verdi, A. Sami, J. Akhondali, F. Khomh, G. Uddin, and A. K. Motlagh, "An empirical study of C++ vulnerabilities in crowd-sourced code examples," 2021, *arXiv:1910.01321*.

[7] MITRE, "Weaknesses in software written in C." Accessed: Oct. 01, 2019. [Online]. Available: https://cwe.mitre.org/data/definitions/658.html

[8] MITRE, "Weaknesses in software written in C++." Accessed: Oct. 01, 2019. [Online]. Available: https://cwe.mitre.org/data/definitions/659.html

[9] WhiteSource, "What are the most secure programming languages?" Accessed: Jan. 04, 2020, 2019. [Online]. Available: https://www.whitesourcesoftware.com/most-secure-programming-languages/

[10] Slashdot, "Which programming language has the most security vulnerabilities?" Accessed: Jan. 04, 2020, 2019. [Online]. Available: https://developers.slashdot.org/story/19/03/25/0322202/which-programming-language-has-the-most-security-vulnerabilities

[11] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 289–305.

[12] TIOBE, "Programming Language C awarded Programming Language of the Year 2019." Accessed: Oct. 01, 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/

[13] L. An, O. Mlouki, F. Khomh, and G. Antoniol, "Stack overflow: A code laundering platform?," in *Proc. IEEE 24th Int. Conf. Softw. Anal. Evol. Reeng.*, 2017, pp. 283–293.

[14] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from StackOverflow: An exploratory study on android apps," *Inf. Softw. Technol.*, vol. 88, pp. 148–158, 2017.

[15] D. Yang, A. Hussain, and C. V. Lopes, "From query to usable code: An analysis of stack overflow code snippets," in *Proc. 13th Int. Conf. Mining Softw. Repositories*, 2016, pp. 391–402.

[16] S. Baltes, L. Dumani, C. Treude, and S. Diehl, "SOTorrent: Reconstructing and analyzing the evolution of stack overflow posts," in *Proc. 15th Int. Conf. Mining Softw. Repositories*, 2018, pp. 319–330.

[17] I. Keivanloo, J. Rilling, and Y. Zou, "Spotting working code examples," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 664–675.

[18] T. Liu and R. Huuck, "Case study: Static security analysis of the android goldfish kernel," in *Proc. Int. Symp. Formal Methods*, 2015, pp. 589–592.

[19] A. Joshi, A. Tewari, V. Kumar, and D. Bordoloi, "Integrating static analysis tools for improving operating system security," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 4, pp. 1251–1258, 2014.

[20] S. V. Yulianto and I. Liem, "Automatic grader for programming assignment using source code analyzer," in *Proc. Int. Conf. Data Softw. Eng.*, 2014, pp. 1–4.

[21] D. J. Worth, C. Greenough, and L. Chin, "A survey of C and C++ software tools for computational science," *Sci. Technol. Facilities Council*, pp. 1–38, 2009.

[22] O. V. Pomorova and D. O. Ivanchyshyn, "Assessment of the source code static analysis effectiveness for security requirements implementation into software developing process," in *Proc. IEEE 7th Int. Conf. Intell. Data Acquisition Adv. Comput. Syst.*, 2013, pp. 640–645.

[23] A. Arusoaie, T. Ciobâcă, V. Crăciun, D. Gavrilut, and D. Lucanu, "A comparison of static analysis tools for vulnerability detection in C/C++ code," in *Proc. Int. Symp. Symbolic Numeric Algorithms Sci. Comput.*, 2017, pp. 161–168.

[24] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, vol. 33, pp. 363–374, 1977.

[25] Q. Chen, L. Bao, L. Li, X. Xia, and L. Cai, "Categorizing and predicting invalid vulnerabilities on common vulnerabilities and exposures," in *Proc. 25th Asia-Pacific Softw. Eng. Conf.*, 2018, pp. 345–354.

[26] MITRE, "2019 CWE top 25 most dangerous software errors." Accessed: Jan. 28, 2021, 2019. [Online]. Available: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

[27] M. Howard, "Improving software security by eliminating the CWE top 25 vulnerabilities," *IEEE Secur. Privacy*, vol. 7, no. 3, pp. 68–71, May/Jun. 2009.

[28] S. Wang, T.-H. Chen, and A. E. Hassan, "How do users revise answers on technical Q&A websites? A case study on stack overflow," *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 1024–1038, Sep. 2020.

[29] A. Bosu, C. S. Corley, D. Heaton, D. Chatterji, J. C. Carver, and N. A. Kraft, "Building reputation in StackOverflow: An empirical investigation," in *Proc. 10th Work. Conf. Mining Softw. Repositories*, 2013, pp. 89–92.

[30] D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steenkiste, and C. Faloutsos, "Analysis of the reputation system and user contributions on a question answering website: StackOverflow," in *Proc. IEEE/ACM Int. Conf. Advances Soc. Netw. Anal. Mining*, 2013, pp. 886–893.

[31] M. Chen, F. Fischer, N. Meng, X. Wang, and J. Grossklags, "How reliable is the crowdsourced knowledge of security implementation?" in *Proc. 41st Int. Conf. Softw. Eng.*, 2019, pp. 536–547.

[32] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in, *Guide to Advanced Empirical Software Engineering*. Berlin, Germany: Springer, 2008, pp. 285–311.

[33] S. Khandelwal, S. K. Sripada, and Y. R. Reddy, "Impact of gamification on code review process: An experimental study," in *Proc. 10th Innov. Softw. Eng. Conf.*, 2017, pp. 122–126.

[34] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: Sentiment analysis of security discussions on GitHub," in *Proc. 11th Work. Conf. Mining Softw. Repositories*, 2014, pp. 348–351.

[35] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *Proc. IEEE Cybersecur. Develop.*, 2017, pp. 22–26.

[36] R. C. Seacord, *Secure Coding in C and C++*. Reading, MA, USA: Addison-Wesley, 2013.

[37] Y. Younan, W. Joosen, F. Piessens, and H. V. den Eynden, "Security of memory allocators for C and C++," *Department of Computer Science, Katholieke Universiteit Leuven, Belgium*, Jul. 2005. [Online]. Available: http://www.fort-knox.org/system/files/CW419.pdf

[38] R. Seacord, "Secure coding in C and C++ of strings and integers," *IEEE Security Privacy*, vol. 4, no. 1, pp. 74–76, Jan./Feb. 2006.

[39] Y. Younan, W. Joosen, F. Piessens, and H. Van den Eynden, "Improving memory management security for C and C++," *Int. J. Secure Softw. Eng.*, vol. 1, no. 2, pp. 57–82, 2010.

[40] H. Mcheick, H. Dhiab, M. Dbouk, and R. Mcheik, "Detecting type errors and secure coding in C/C++ applications," in *Proc. ACS/IEEE Int. Conf. Comput. Syst. Appl.*, 2010, pp. 1–9.

[41] W. Dietz, P. Li, J. Regehr, and V. Adve, "Understanding integer overflow in C/C++," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 1, pp. 2:1–2:29, Dec. 2015.

[42] S. M. Alnaeli, M. Sarnowski, M. S. Aman, K. Yelamarthi, A. Abdelgawad, and H. Jiang, "On the evolution of mobile computing software systems and C/C++ vulnerable code: Empirical investigation," in *Proc. IEEE 7th Annu. Ubiquitous Comput. Electron. Mobile Commun. Conf.*, 2016, pp. 1–7.

[43] S. M. Alnaeli, M. Sarnowski, M. S. Aman, A. Abdelgawad, and K. Yelamarthi, "Vulnerable C/C++ code usage in IoT software systems," in *Proc. IEEE 3rd World Forum Internet of Things*, 2016, pp. 348–352.

[44] J. Yang, L. Tan, J. Peyton, and K. A. Duer, "Towards better utilizing static application security testing," in *Proc. 41st Int. Conf. Softw. Eng.: Softw. Eng. Practice*, 2019, pp. 51–60.

[45] C. Treude and M. P. Robillard, "Understanding stack overflow code fragments," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 509–513.

[46] U. Campos, G. Smethurst, J. A. P. Moraes, R. Bonifácio, and G. Pinto, "Mining rule violations in JavaScript code snippets," in *Proc. 16th Int. Conf. Mining Softw. Repositories*, 2019, pp. 195–199.

[47] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 910–924, Sep. 2016.

[48] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in stack overflow," *Empir. Softw. Eng.*, vol. 19, no. 3, pp. 619–654, Jun. 2014.

[49] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Proc. IEEE Symp. Secur. Privacy*, 2016, pp. 289–305.

[50] T. Lopez, T. T. Tun, A. Bandara, M. Levine, B. Nuseibeh, and H. Sharp, "An investigation of security conversations in stack overflow: Perceptions of security and community involvement," in *Proc. 1st Int. Workshop Secur. Awareness Des. Deployment*, 2018, pp. 26–32.

Haoxiang Zhang is a senior researcher at the Centre for Software Excellence at Huawei, Canada. His research interests include empirical software engineering, mining software repositories, and intelligent software analytics. He received the PhD degree in computer science from Queen's University, Canada and the second PhD degree in physics and the MSc degree in electrical engineering from Lehigh University, and obtained his BSc in Physics from the University of Science and Technology of China. For more information, please visit https://haoxianghz.github.io/.

Shaowei Wang received the BSc degree from Zhejiang University, Hangzhou, China, and the PhD degree from Singapore Management University, Singapore. He is an assistant professor with the Department of Computer Science, University of Manitoba. His research interests include software engineering, machine learning, data analytics for software engineering, automated debugging, and secure software development. He is one of four recipients of the 2018 Distinguished Reviewer Award for the Springer EMSE (SE's highest impact journal). For more information, please visit https://sites.google.com/site/wswshaoweiwang/.

Heng Li received the BEng degree from Sun Yat-sen University, Guangzhou, China, the MSc degree from Fudan University, Shanghai, China, and the PhD degree from the School of Computing, Queen's University, Kingston, Canada. He is an assistant professor with the Department of Computer Engineering and Software Engineering, Polytechnique Montreal, Montreal, Canada, where he leads the Maintenance, Operations and Observation of Software with Intelligence (MOOSE) Lab. He also worked with Synopsys as a software engineer for two years and worked with BlackBerry as a software performance engineer for another two years. His research interests lie within software engineering, in particular, software observability, intelligent operations of software systems, software log mining, software performance engineering, and mining software repositories. More information at: https://www.hengli.org.

Tse-Hsun Chen received the BSc degree from the University of British Columbia, Vancouver, Canada, and the MSc and PhD degrees from Queen's University, Kingston, Canada. He is an assistant professor with the Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada. He leads the Software Performance, Analysis, and Reliability (SPEAR) Lab, which focuses on conducting research on performance engineering, program analysis, log analysis, production debugging, and mining software repositories. His work has been published in flagship conferences and journals such as ICSE, FSE, *IEEE Transactions on Software Engineering*, *Empirical Software Engineering*, and MSR. He serves regularly as a program committee member of international conferences in the field of software engineering, such as ASE, ICSME, SANER, and ICPC, and he is a regular reviewer for software engineering journals such as the *Journal of Systems and Software*, *Empirical Software Engineering*, and *IEEE Transactions on Software Engineering*. Besides his academic career, he also worked as a software performance engineer with BlackBerry for more than four years. Early tools developed by him were integrated into industrial practice for ensuring the quality of large-scale enterprise systems. For more information, please visit http://petertsehsun.github.io/.

Ahmed E. Hassan (Fellow, IEEE) received the PhD degree in computer science from the University of Waterloo, Waterloo, Canada. He is an ACM SIGSOFT influential educator, an NSERC steacie fellow, the Canada research chair (CRC) in software analytics, and the NSERC/BlackBerry software engineering chair with the School of Computing, Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves/d on the editorial boards of *IEEE Transactions on Software Engineering*, *Springer Journal of Empirical Software Engineering*, and *PeerJ Computer Science*. For more information, please visit http://sail.cs.queensu.ca/.