Stack Overflow Meets Replication: Security Research Amid Evolving Code Snippets

Alfusainey Jallow[§*], Sven Bugiel[§]

[§] CISPA Helmholtz Center for Information Security, [*] Saarland University

Abstract

We study the impact of Stack Overflow code evolution on the stability of prior research findings derived from Stack Overflow data and provide recommendations for future studies. We systematically reviewed papers published between 2005–2023 to identify key aspects of Stack Overflow that can affect study results, such as the language or context of code snippets. Our analysis reveals that certain aspects are non-stationary over time, which could lead to different conclusions if experiments are repeated at different times. We replicated six studies using a more recent dataset to demonstrate this risk. Our findings show that four papers produced significantly different results than the original findings, preventing the same conclusions from being drawn with a newer dataset version. Consequently, we recommend treating Stack Overflow as a time-series data source to provide context for interpreting cross-sectional research conclusions.

1 Introduction

In recent years, several security-focused studies [5,6,19,22,25,27,37,56,57,61,62,68,78,84] have examined Stack Overflow to analyze the security of shared code on the platform, develop tools for secure code reuse, or use it as a proxy for studying developer behavior. This research is fostered by the quarterly releases of a dataset containing all content created on the Stack Exchange Inc. platform since its launch in 2008. Right now, about 60 million posts (24 million questions and 35.8 million answers) [39] containing over 91 million comments are publicly available for analysis.

However, Stack Overflow code and content evolves as the community adds and updates snippets [13,37,84]. This code evolution has already been shown to negatively affect developers who reuse a specific snippet version from Stack Overflow without tracking updates for security fixes [43].

Like developers, security researchers may also study the content of the Stack Overflow data set only using the current version, i.e., *cross-sectional* studies. While cross-sectional studies provide valuable and novel insights into the underlying data, other disciplines [14, 32, 81] and also software engineering research [74] suggest that complementing these insights with longitudinal trend and time series analysis provides a better context for interpreting findings. Transferring these lessons to Stack Overflow code evolution raises questions about how this evolution affects cross-sectional research findings based on particular dataset versions and what lessons can be learned for future studies using Stack Overflow data. While a shift in research results due to a shift in the data is intuitive, this phenomenon has not been systematically studied before for Stack Overflow-based data-driven research. To offer new insights into this issue, this paper aims to address the following meta-research questions:

MQ1: *Which aspects of Stack Overflow affect the results of prior research?*
MQ2: *How much do Stack Overflow code snippets and surrounding context evolve?*
MQ3: *How would the results of prior research differ if replicated on a newer Stack Overflow dataset version?*

To answer our research questions, we surveyed the literature for papers studying the security properties of code snippets on Stack Overflow. This yielded 42 highly relevant papers, which we systematized according to the Stack Overflow aspects their methods relied on (e.g., programming language or the context of snippets). This systematization shows that the targeted programming language may affect the stability of results over time and that most works leverage some form of code classification, whose results can be immediately affected by code revisions (*MQ1*). Further, we conducted a time series analysis to understand how code snippets and security- and privacy-related discussions on Stack Overflow evolve. Our data shows that programming languages trend differently regarding their overall number of added snippets and their ratio of security-relevant edits. Moreover, we found that the fraction of security-relevant comments on Stack Overflow steadily increased. As a result, studies focusing on particular programming languages will likely find a different landscape when

conducted at various points in time (*MQ2*). Together, these two insights provide an intuition about how prior research results may shift in light of the evolution of Stack Overflow content. To provide concrete evidence for the impact of this evolution on research results (*MQ3*), we conducted *six* replication studies of prior work [22, 25, 27, 37, 62, 84] using a more recent data set version. We find that the results of multiple works shift over time [22, 37, 62, 84]. For example, the landscape of CWEs in C/C++ code snippets [84] has significantly shifted, and Stack Overflow now contains proportionally more vulnerable snippets with different ratios for CWE types. Only the results of two papers on crypto API misuse in Java snippets [25, 27] remained stable (see [45]). We postulate that this may be due to the particular niche topic that requires domain experts to identify and fix such vulnerabilities. Based on our replication studies, we offer advice for future research involving Stack Overflow data. A shift in results does *not* mean that the results are invalid but missing context. We recommend that researchers consider data on Stack Overflow as time-series data and discuss their results as a trend model rather than a cross-sectional analysis—taking inspiration from methods in economics, environmental science, or medical studies. This approach provides a more meaningful context for the results, allowing us to determine whether the observed issues are short-term trends or persistent systemic issues.

2 Background & Motivation

Developers seeking advice for a programming problem can create a question post on Stack Overflow, which other developers can answer. The question and all the answers are called *posts*, each with a unique *identifier*. Developers can also *comment* on posted code snippets through Stack Overflow's commenting feature. Comments made by other developers are known to induce updates to posts [70] or raise bug reports and point out security vulnerabilities [37, 43, 84].

Stack Exchange Inc. has a quarterly release cycle of all data created on the platform since its inception in 2008. This recurring release allows researchers to tap into this rich data source (e.g., [22, 25, 27, 37, 43, 61, 62, 84]). Unfortunately, the Stack Exchange dataset only provides versioning at the level of whole posts and not at the level of individual text and code snippets, which makes it non-trivial to track and analyze changes to individual code snippets contained in a post. The SOTorrent open dataset by Baltes et al. [13] is based on the official Stack Exchange data and provides version control at the level of individual text and code snippets. For this reason, it is popular among researchers studying Stack Overflow.

Motivating Example Figure 1 depicts an example snippet [73] from the 350+ examples of the results of Zhang et al. [84] that were affected by Stack Overflow evolution. Zhang et al. used the 12/2018 version of the SOTorrent dataset to study whether code revisions improve security. They identified two CWEs in this snippet: two instances of CWE-775 and one of CWE-401, all introduced in the first version in January 2013. By December 2018, this snippet had never been edited.Therefore, the authors correctly labeled the snippet as insecure and concluded that the snippet *had never been revised* to address the three CWE instances. By July 2020, the snippet had been revised to fix both CWE-775 instances. This revision, posted *after* the authors sampled their data, changed the snippet's status to *improved*.

Figure 1: First version of the snippet (left) labeled as *insecure and unmodified* with three CWE instances. The 2^{nd} version (right) shows both instances of CWE-775 fixed on July 4, changing the snippet's status to *improved*.

The example shows that a code snippet can undergo edits throughout its lifespan, causing its security status to change over time. Consequently, code snippets might be insecure at a given time but secure at a future time or vice-versa. Therefore, it seems prima facie intuitive for researchers studying Stack Overflow snippets to conduct measurements at multiple points in time using different dataset versions to account for these fluctuations. Integrating these evolving trends and changes in snippets may enhance the robustness of research studies.

3 Systematization of Relevant Works

To understand how prior work may be affected by Stack Overflow evolution (**MQ1**), we systematize studies that investigated the security properties of Stack Overflow code snippets. First, we systematically surveyed the literature for relevant works (§3.1). Following this, we created a taxonomy of the methodologies by those studies for analyzing Stack Overflow datasets (§3.2). Two researchers conducted this systematization and decided on the criteria for the taxonomy.

3.1 Literature Search

We conducted a systematic literature review following the guidelines by Kitchenham and Charters [47].

Inclusion and Exclusion Criteria. The inclusion criteria are that the study: *IC1.* must focus on the Stack Overflow

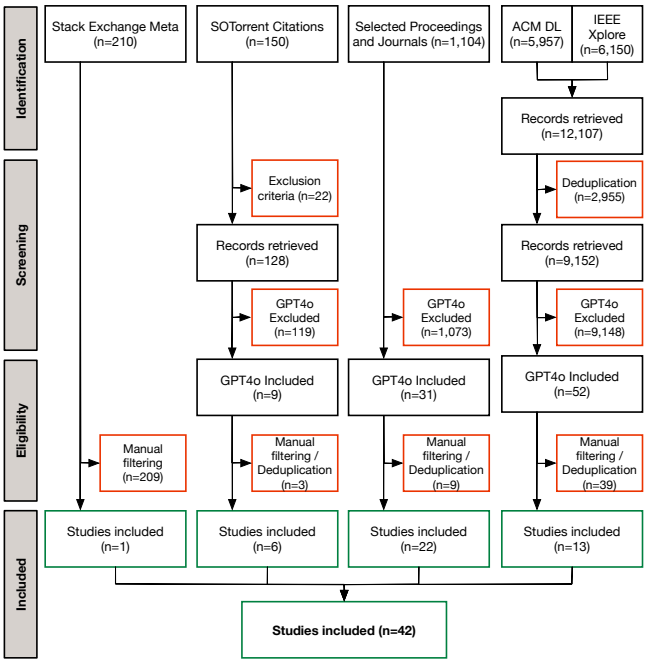


Figure 2: PRISMA flowchart of our literature review

Listing 1: Search terms for literature search

```
" stackoverflow " OR "stack overflow" OR "crowd knowledge" OR
"crowdsource knowledge" OR "crowd–source knowledge" OR "Q&A websites" OR
"Q&A sites" OR "social Q&A websites" OR "online Q&A communities" OR
"community question answering" OR "knowledge sharing" OR "crowdsourcing" OR
"knowledge–sharing" OR "Q&A Forums" OR "Online Code Snippets"
```

website. *IC2.* must examine code snippets on Stack Overflow. *IC3.* must analyze the security of code snippets or identify and address bugs or faults in snippets. *IC4.* must be published between 01/2005 and 12/2023. *IC5.* is a journal paper or in conference proceedings that are DBLP-indexed. The exclusion criteria are: *EC1.* Studies focusing only on code reuse from Stack Overflow without studying the security of or identifying bugs or faults in snippets. *EC2.* Publications outside the date range in *IC4*. *EC3.* Systematic reviews or meta-analyses, as only primary studies are considered. *EC4.* Non-peer-reviewed studies (e.g., technical reports).

Search Strategy. Figure 2 depicts the flow of our systematic literature review (SLR). We manually reviewed the titles and abstracts of studies listed on the Meta Exchange thread [72] listing academic papers that use Stack Exchange datasets. This yielded one relevant paper. Next, we retrieved all studies that cited the SOTorrent dataset [13], a widely used dataset for studying Stack Overflow. Of these 150 studies, 128 fit the inclusion criteria. Following the recommendations by Kitchenham and Charters [47], we automated part of the SLR. We employed OpenAI's GPT-4o model to scan abstracts and identify studies focusing on the security of Stack Overflow code snippets. A preliminary evaluation of GPT-4o showed that it performs well in classifying studies based on our inclusion and exclusion criteria (details in [45]). We use GPT-4o to screen non-relevant studies based on a better contextual understanding of security concepts than keyword-based filtering. Papers identified as relevant by GPT-4o are manually verified through full-text analysis. With the help of GPT-4o, we identified six additional relevant studies among the 128 candidates. Based on the reviewed studies, we created a list of search terms to identify *all* research studies related to Stack Overflow, irrespective of whether they analyzed code snippets or investigated their security. The search terms are listed in Listing 1. We used these terms to search the proceedings of 29 conferences and the volumes of 3 journals from security & privacy, HCI, and software engineering venues (full list in [45]). We identified 1,104 matching studies across all venues. Using GPT-4o, we analyzed their abstracts, resulting in 22 confirmed relevant studies. We conducted additional searches in IEEE Xplore and ACM Digital Library using the same search terms. After deduplication, this resulted in 9,152 unique publications, which we retrieved from the two libraries. Using GPT-4o, we confirmed an additional 13 studies that met our criteria, totaling 42 relevant studies.

3.2 Comparison Criteria

Our literature search yielded 42 relevant studies. We reviewed the full text of each study and its artifacts to develop criteria for comparison of different works. We found nine criteria, explained below, and Table 1 compares the 42 considered studies based on these criteria. Criteria **D1**–**D5** define dependencies on Stack Overflow data (used in §4), where criteria **R1**–**R4** are relevant for our replication studies (in §6). For presentation, we adopt the pictogram-based visualization style common in *Systematization of Knowledge* papers [21].

D1. Programming Languages: Lists the programming language(s) of code snippets investigated in a study. If a study selects snippets independently of language, we assign ✪.

D2. Code Scanners: *Off-the-shelve* scanners (✔) or *custom* code classifiers (✪) can be used to find security weaknesses or general code quality issues in code snippets. For custom solutions, we list the techniques for finding security flaws, e.g., static analysis, machine learning, or graph query analysis. We assign ✘ for studies detecting security issues manually.

D3. Code Evolution: Indicates whether a study considered (✔) code evolution in their methodology or not (✘).

D4. Surrounding Context: Comments and post descriptions are natural language text that provides context around code snippets. We assign ✔ if a paper utilizes surrounding context to classify code snippets (e.g., using NLP); otherwise, ✘.

D5. Sample Size Filter: Studies may exclude code snippets based on specific criteria. For example, a study examining cryptographic API usage in Java might filter out snippets that

Table 1: Comparison of security-focused studies on Stack Overflow. We replicated the highlighted studies.

		Dataset Snapshot	D1. Prog. Languages	D2. Code Scanners	D3. Code Evolution	D4. Surrounding Context	D5. Sample Size Filter	R1. Artifact Availability	R2. Language Detection	R3. Code Reuse Detect.	R4. Human-Centered
Zhang et al.	[84]	12/2018	C	✔	✔	✘	✔	✘	✪, M	✘	✘
Hong et al.	[37]	12/2020	C	✪, N	✔	✔	✔	○	✔	✪, S	✘
Fischer et al.	[27]	03/2018	J	✪, M	✘	✘	✔	◐	N/A	✘	✘
Fischer et al.	[25]	03/2016	J	✪, M	✘	✘	✔	◐	N/A	✪, P	✘
Rahman et al.	[62]	12/2018	P	✪, SM	✘	✘	✔	◑	✔	✘	✘
Campos et al.	[22]	12/2018	JS	✔	✘	✘	✔	✪	✔	✔	✘
Verdi et al.	[78]	09/2018	C	✘	✘	✘	✔	◐	✔	✔	✘
Selvaraj et al.	[68]	01/2022	C	✔	✔	✘	✔	◐	✪, M	✘	✘
Acar et al.	[6]	10/2015	J	✘	✘	✘	✘	✘	N/A	✘	✔
Chen et al.	[19]	?/2018	J	✘	✘	✘	✔	✘	N/A	✘	✘
Meng et al.	[57]	08/2017	J	✘	✘	✔	✔	◐	✔	✘	✘
Ragkhitwets	[61]	01/2016	J	✘	✔	✘	✔	◐	✔	✪,SI,CC	✔
Bai et al.	[11]	N/A	J	✘	✘	✘	✔	✘	N/A	✪,M	✔
Bagherzadeh et al.	[10]	N/A	J,S	✘	✘	✔	✔	◐	✔	✘	✘
Chen et al.	[18]	?/2018	J	✪,M	✘	✘	✔	✘	N/A	✘	✘
Zhang et al.	[85]	10/2016	J	✪,P	✘	✘	✔	✘	✔	✔,CC	✘
Rahman et al.	[63]	08/2021	J	✘	✘	✘	✔	✘	✔	✔,CC	✔
Reinhardt et al.	[64]	N/A	J	✔	✘	✘	✔	✘	✔	✔,CC	✘
Licorish et al.	[49]	?/2016	J	✔	✘	✔	✘	✘	✔	✘	✘
Schmidt et al.	[66]	03/2022	JS,P	✘	✘	✔	✘	◑	✔	✘	✘
Yi Liu et al.	[51]	N/A	J	✪,M	✘	✔	✘	✘	✔	✘	✘
Ren et al.	[65]	03/2019	J	✪,M	✘	✔	✔	✘	✔	✘	✔
Licorish et al.	[50]	?/2016	J	✔	✘	✘	✔	✘	✔	✘	✘
Rangeet Pan	[60]	N/A	P	✘	✘	✔	✔	✘	✔	✘	✘
Ye et al.	[83]	N/A	J	✪,M	✘	✘	✔	✘	✔	✘	✘
Chen et al.	[17]	N/A	J	✪,M	✘	✔	✔	✘	✔	✔,CX	✔
Zhang et al.	[86]	N/A	P	✘	✘	✘	✔	◑	N/A	✘	✘
Alhanahnah et al.	[8]	N/A	J	✔	✘	✔	✔	◑	N/A	✘	✘
Imai et al.	[38]	N/A	J	✔	✘	✘	✘	✘	✪,R	✪,Se	✘
Fischer et al.	[26]	03/2018	J	✪,M	✘	✘	✔	✘	N/A	✘	✔
Almeida et al.	[9]	N/A	JS	✔	✘	✘	✔	✘	N/A	✘	✘
Islam et al.	[42]	N/A	P	✘	✘	✘	✔	◑	N/A	✘	✘
Mahajan et al.	[54]	03/2019	J	✪,A	✘	✘	✔	◑	✔	✘	✔
Mahajan et al.	[55]	03/2019	J	✔	✘	✘	✔	◑	N/A	✘	✔
Yadavally et al.	[82]	N/A	J,C	✘	✘	✘	✔	◑	✪,S	✘	✘
Firouzi et al.	[24]	09/2018	C#	✘	✘	✘	✔	✘	✔	✘	✘
Ghanbari et al.	[30]	09/2018	P	✪,F	✘	✘	✔	✘	✔	✘	✘
Gao et al.	[29]	N/A	J	✔	✘	✘	✔	✘	N/A	✘	✘
M. Chakraborty	[16]	01/2021	P	✘	✘	✘	✔	✘	N/A	✘	✘
Moghadam et al.	[59]	N/A	J	✘	✘	✘	✔	✘	✔	✘	✘
Madsen et al.	[53]	N/A	JS	✪,S	✘	✔	✔	✘	✔	✘	✘
Jhoo et al.	[46]	N/A	P	✪,S	✘	✘	✔	✘	✔	✘	✘

D1: J = Java, C = C/C++, JS = JavaScript, P = Python, S = Scala, P = PHP
D2: SM = String matching, M = Machine learning, N = NLP, R = Regular Expression, A = Abstract Program Graph, P = Pattern Recognition, F = Fault Localization, S = Static analysis
R2: M = Machine Learning, R = Regular Expression, S = Static analysis
R3: S = SourcererCC, SI = Simian, CC = CCFinder, CX = CCFinderX, P = PDG, Se = SeByte, M = MOSS

do not use those APIs, or a study focusing on the effects of code revisions might exclude single-version snippets. We assign ✘ if a study includes all snippets without filtering; otherwise, we assign ✔.

R1. Artifact Availability: Code and data artifacts are important for revisiting and replicating prior findings; here, they are used to compare results on different versions of Stack Overflow datasets without undergoing the tedious (and error-prone) effort of re-implementing prior approaches. If paper artifacts (either code, data, or both) are unavailable, we assign ✘. We distinguish between artifacts that are *fully* or *partially* available. An artifact is *fully functional available* (✪) if both data and code are available and, most importantly, the code is free of bugs and can be used directly without changes to the code base. We assign ● if the code of a fully available artifact has bugs that require significant effort to fix. An artifact is *partially* available if only code (◑) or only data (◐) is available. If the code of a partially available artifact requires significant bug fixing, we assign ○.

R2. Language Detection: The Stack Exchange and SOTorrent datasets do not state the programming languages of code snippets, requiring researchers to determine these. Researchers can rely on other data (✔) like *tags* of posts, or they can employ code analysis tools (✪). When a tool is used, we indicate the underlying technique (e.g., machine learning or static analysis) used for language detection. If the authors did not mention their detection technique, we assign **N/A**.

R3. Code Reuse Detection: Several studies indicate that developers reuse code from Stack Overflow, sometimes attributing the copied code snippets with their URL [12]. This criterion differentiates between studies using attribution (✔) and studies using tools (✪), e.g., clone detection, to identify code reuse. If a tool is used, we give the name of the tool or technique. We assign ✘ if code reuse is not considered.

R4. Human-Centered Research: Differentiates human-centered studies of Stack Overflow data involving software developers as participants. If the study's methodology includes developer participation, we assign ✔; otherwise, we assign a ✘ for non-user-driven studies.

4 Relevance of Stack Overflow Evolution

Table 1 lists the relevant works from our literature review. Following, we elaborate on the criteria for these papers (**D1**–**D5**) and how these can be affected by content evolution on Stack Overflow. For space reasons, we restrict our explanations to the six highlighted studies in Table 1, with explanations of the remaining studies available in our extended version [45].

The work by Zhang et al. [84] investigated whether code revisions on Stack Overflow are associated with improving or worsening the security of code snippets (**D3**: ✔). The study focused on *C* and *C++* code snippets (**D1**: C/C++) with at least 5 LoC (**D5**: ✔). The *CppCheck* static analysis tool was

used to detect security weaknesses (**D2**: ✔, **D4**: ✘).

The *DICOS* tool by Hong et al. [37] analyzes the change history of C/C++ code snippets (**D1**: C/C++; **D3**: ✔) to discover insecure code snippets by looking for changes in security-sensitive APIs, control flow information, and/or security-related keywords in the surrounding context (**D2**: ✪, N; **D4**: ✔). However, the authors only considered the first and last revision of each snippet (**D5**: ✔).

Fischer et al. [25] investigated the reuse of insecure Java code snippets in Android apps (**D1**: Java). They employed a machine learning classifier to identify snippets with insecure usage of crypto APIs (**D2**: ✪, M; **D5**: ✔). The authors considered neither code evolution (**D3**: ✘) nor surrounding context in their methodology (**D4**: ✘).

Follow-up work by Fischer et al. [27] relied on flagging Java snippets with crypto API misuse and suggesting more secure snippets (**D1**: Java; **D2**: ✪, D; **D5**: ✔). As in their preceding work, the authors considered neither code evolution (**D3**: ✘) nor context in their methodology (**D4**: ✘).

Rahman et al. [62] analyzed Python code snippets (**D1**: Python) to identify insecure coding practices using string matching (**D2**: ✪, SM). They focused on snippets from answers attributed in GitHub project (**D5**: ✔) but did not consider code evolution or context (**D3:** ✘, **D4**: ✘).

Campos et al. [22] studied JavaScript code snippets (**D1**: JavaScript) to identify rule violations using ESLint, a JavaScript linter (**D2:** ✔). They relied exclusively on ESLint to detect security or code quality issues without considering the surrounding context (**D4**: ✘) or the evolution of snippets (**D3**: ✘). Further, they focused only on snippets with a minimum of 10 lines of code (**D5**: ✔).

MQ1: What aspects of Stack Overflow affect the results of prior research? Criteria **D1**–**D5** in Table 1 show that programming language-specific trends and evolution may affect the stability of results over time. Further, most of these works leverage some form of code classification, making their results susceptible to changes as the code evolves. Only four works rely upon code evolution in their methodology, showing that ongoing evolution beyond the study's timeframe can influence their results. Additionally, twelve studies focus on the context surrounding code snippets, so any changes in context, like the addition of security-relevant comments, could also impact their findings.

5 Evolution of Stack Overflow

Based on the identified aspects of Stack Overflow that *may* affect prior research (MQ1), we now look at the global evolution of the programming languages and security-relevant contexts on Stack Overflow. Measuring the security of code snippets directly, however, is a challenging task and requires dedicated methodologies (e.g., [25, 27, 37, 84]). We address the evolution of code snippet security through case studies in §6 and focus here on programming languages and security-relevant edits and comments.

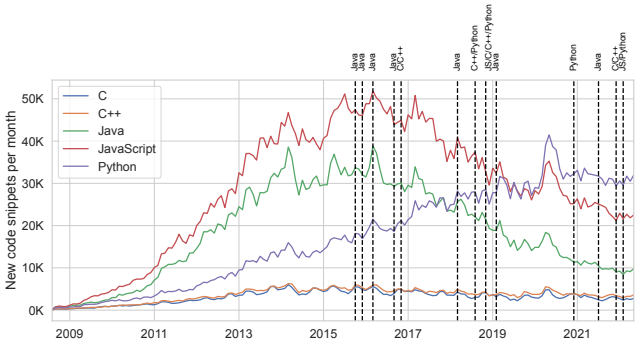


Figure 3: Added code snippets on Stack Overflow per month. Dashed lines indicate data collection points of snippets by the works in our systematization (if known; see Table 1).

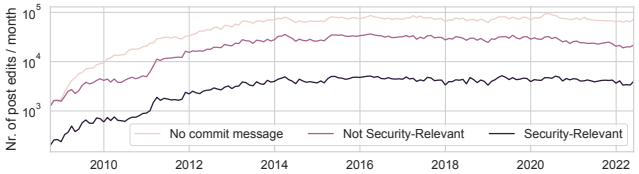
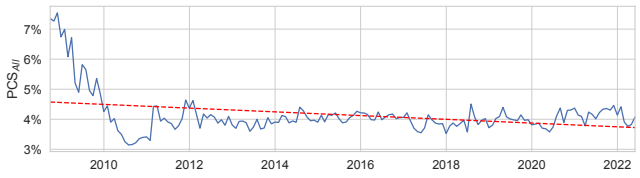


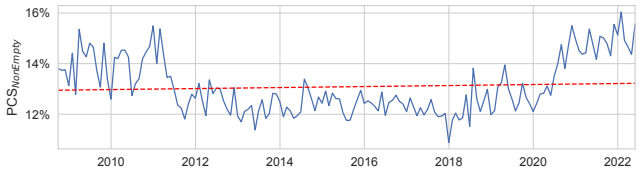
Figure 4: Number of monthly (30-day interval) post edits categorized by their security relevance.

Programming Languages Figure 3 depicts the monthly addition of new code snippets in the most considered programming languages in the works from Table 1. The data shows that while C/C++ is relatively stable over time but at a comparatively low rate, Java and JavaScript peaked between 2013 and 2018 and have been shrinking on Stack Overflow since 2018. We indicate with dashed lines when the papers from our systematization sampled their data (if known).

Security-Relevant Edits and Comments We found that the average code snippet on Stack Overflow receives 1.66 edits (max = 754, $P_{75} = 2$, $P_{99} = 6$). A breakdown of average code snippet edits per language is provided in the extended version of this paper [45]. Figure 4 shows the number of monthly *post* edits on Stack Overflow between 09/2008–06/2022, broken down by their security relevance and commit message. To identify security-relevant edits, we apply the publicly shared NLP-based classifier by Jallow et al. [1, 43] on the commit messages of the edits. We found 9,443,509 code edits without a commit message, 3,731,935 non-security-relevant commit messages, and 549,863 commit messages indicating security relevance. Our data shows that 514,666 answer posts have



(a) Including empty commit messages.



(b) Excluding empty commit messages.

Figure 5: Percentage of security-relevant commits (PSC) in monthly intervals. Dashed lines are fitted linear regressions.

received at least one security-relevant commit (max = 27). We calculate the percentage of security-relevant commits (PSC):

$$PSC = \frac{\text{Number of security-relevant commits} * 100}{\text{Total number of commits}}$$

On June 20, 2020, Stack Overflow changed to CommonMark [79] and adjusted all 338,622 nonconforming posts with automated edits. For data sanity, we exclude these script-generated edits from the PSC calculation.

Figure 5 depicts the PSC for all code edits on Stack Overflow, where Figure 5a counts empty commit messages to the total number of commits and Figure 5b excludes them. The monthly average PSC is $PSC_{All} = 4.2\% \pm 0.1$ and $PSC_{NonEmpty} = 13.1\% \pm 0.2$ ($CI = 95\%$). We use the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test [48] to test for stationarity of the PSC over the long term. Since KPSS is known to exhibit a high rate of Type-I errors, indicating non-stationary too often, we also test the PSC's non-stationary with the Augmented Dickey Fuller (ADF) test [28]. Our extended version [45] details the results and illustrates the corresponding PSC. In summary, when focusing on non-empty commit messages, we find that the overall PSC is trend stationary (KPSS $p > 0.05$; ADF $p > 0.05$), the PSC for C code snippets is difference stationary (KPSS $p > 0.05$; ADF $p < 0.001$) with an increasing PSC (i.e., the difference between data points is stationary and the PSC has a linear upward trend over time), and the PSC for JavaScript snippets is non-stationary (KPSS $p < 0.05$; ADF $p > 0.05$). All but C++ and Python edits show a stationary PSC when considering non-empty commit messages.

Regarding comments, we found that 3,991,533 (8.1%) of 49,087,103 comments were security-relevant. These relevant comments are for 3,128,208 posts, of which 98,139 also received a security-relevant commit message. The mean percentage of security-relevant comments $PSC_{Comments}$ is 7.78 ± 0.143, where posts with C/C++ snippets have a significantly higher mean $PSC_{Comments}$ between 11–12% (refer to [45] for further details). Further, KPSS and ADF tests show that the overall $PSC_{Comments}$ is difference stationary and that the $PSC_{Comments}$ for C/C++, Java, JavaScript, and Python posts is non-stationary (refer to [45] for further details). A fitted linear regression confirms this increasing trend of $PSC_{Comments}$ ($R^2 = 0.93$, $p < 0.001$ for all comments). A potential explanation for this development could be the decreasing number of new code snippets (e.g., because simple questions are now posted to GenAI tools) and a continued community effort to curate the Stack Overflow content.

MQ2: How much do Stack Overflow code snippets and surrounding context evolve? Our data shows that programming languages trend differently in their overall number of added snippets and their ratio of security-relevant edits. Thus, studies focusing on particular languages will likely find a different landscape when conducted at different times. Further, many comments raised security-relevant issues but were largely not on posts that received a security-relevant edit. Over time, the ratio of security-relevant comments steadily increased, indicating that the community strives to improve content quality.

6 Replication Case Studies

We present replication case studies to answer **MQ3** whether the findings of prior research relying on specific versions of the Stack Overflow dataset change due to evolution. In contrast to §4 and §5, we aim to find concrete evidence for the impact of Stack Overflow evolution on research results. We detail four replication case studies [22, 37, 62, 84] and present two more [25, 27] in our extended version [45].

Excluded papers. As shown in Table 1, we focused on six papers for replication and excluded the others. In 16 studies [10, 11, 16, 24, 42, 59, 60, 63, 66, 82, 86], the detection of security weaknesses and bugs in code snippets was not automated; instead, these studies relied on manual processes to label and identify security issues and/or bugs in code snippets. This approach typically involved human annotators reviewing and classifying the code for potential weaknesses, which is hard to compare in a replication study with different human evaluators. On the other hand, three studies [17, 26, 65] leveraged machine learning techniques to automate the detection of security vulnerabilities in code snippets. While these studies employed advanced algorithms to facilitate automated analysis, they failed to release the underlying source code and datasets, particularly the training data used for training their models, making replication hard. Additionally, nine studies [8, 9, 18, 29, 46, 51, 53, 64, 83] did not specify which version

Table 2: Results by Zhang et al. (cf. Table 1 in [84]) versus our replication study using SOTorrent22 and Cppcheck v2.13.

Based on SOTorrent18 (Original)				Based on SOTorrent22 (Replication)			
	Answer #	*Code Snippet #*	*Code Version #*		*Answer #*	*Code Snippet #*	*Code Version #*
SOTorrent	867,734	1,561,550	1,833,449	SOTorrent	1,096,380	1,944,378	2,340,975
LOC >= 5	527,932	724,784	919,947	LOC >= 5	695,326	938,643	1,234,443
Guesslang	490,778	646,716	826,520	Cppcheck 2.13	323,321	388,749	507,997
$Code_w$	**11,235**	**11,748**	**14,934**	$Code_w$	**28,521**	**30,254**	**38,248**

of the dataset they used to collect code snippets from Stack Overflow. Lastly, nine studies [6, 11, 17, 26, 54, 55, 61, 63, 65] were excluded in favor of non-user-driven studies.

Datasets The selected papers investigated two kinds of Stack Overflow datasets. First, Fischer et al. [25, 27] used the official data dump provided by Stack Exchange, Inc [71]. Here, we perform a replication study using the September 2023 version (denoted *StackExchange23*). Second, Zhang et al. [84], Hong et al. [37], Campos et al. [22] and Rahman et al. [62] used the *SOTorrent* dataset by Baltes et al. [13]. Here, we perform a replication study using a newer dataset version. The authors of the SOTorrent dataset stopped providing new releases in December 2020. Fortunately, the tool used to make new releases of the dataset is open-source [67], allowing us to create a new release of SOTorrent (denoted *SOTorrent22*) based on the *June 2022* version of the Stack Exchange dataset.

Notation Colored text is used to distinguish between results from the original studies and those from our replication.

6.1 Case Study 1: C/C++ Code Weaknesses

Zhang et al. [84] studied whether revisions to C/C++ snippets increase or decrease the snippets' security. Their work addressed the following questions: **RQ1**: *What are the types of code weaknesses that are detected in C/C++ code snippets on Stack Overflow?* **RQ2**: *How do code with weaknesses evolve through revisions?* **RQ3**: *What are the characteristics of the users who contributed code with weaknesses?*

6.1.1 Original Methodology

The authors employed a data-driven approach (**R4**: ✘), focusing exclusively on answer posts from the *SOTorrent18* dataset (released in 12/2018), without considering code reused from Stack Overflow (**R3**: ✘). The left-hand side of Table 2 shows the originally collected data. The authors extracted 867,734 answers containing 1,561,550 code snippets with C/C++ tags with 1,833,449 versions. From this data set, they filtered all snippets with less than five LoC. This resulted in 724,784 code snippets (with 919,947 versions) from 527,932 answers. However, the authors noticed that using *tags* alone is insufficient to determine the language of code snippets, i.e., not all snippets contain valid C/C++ code. Using the *Guesslang* machine learning classifier [31] they filtered non-C/C++ code snippets (**R2**: ✪, **M**). This resulted in 646,716 code snippets (having 826,520 versions) from 490,778 answers. In a final step, the authors leveraged the *CppCheck* static analysis tool to scan all 826,520 versions to detect security weaknesses. We reuse the terms by the authors to denote code snippets, snippet versions, and answers with security weaknesses as $Code_w$, $Version_w$, and $Answer_w$, respectively. The authors' final dataset to answer their research questions are $Version_w = 14,934$ from $Code_w = 11,748$ in $Answer_w = 11,235$.

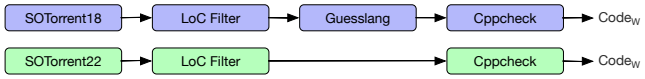


Figure 6: Comparison of Zhang et al.'s [84] methodology with the approach used in our replication study.

6.1.2 Re-Implementation

The authors did not publicly release their source and data artifacts (**R1**: ✘) but provided us on request with a CSV file of the $Answer_w = 11,235$. The unavailability of the code artifact forced us to re-implement their methodology for replication.

An exact re-implementation of Zhang et al.'s [84] methodology was impossible. The Guesslang version used by the authors is outdated and unavailable. Replacing the old version with the newest version, v2.2.1, yielded significantly different numbers for the same *SOTorrent18* data set (490,778 vs.249,451). Additionally, we noticed that Guesslang v2.2.1 misclassified 4,901 C/C++ answers (out of $Answer_w = 11,235$) from the authors' results as Java. Given the substantial differences of Guesslang v2.2.1, we devised an alternative methodology that skips Guesslang (see Figure 6). We relied solely on Cppcheck to identify valid C/C++ snippets and detect security weaknesses. Cppcheck attempts to compile code snippets, failing if the snippets are not valid C/C++ code. Thus, Cppcheck formed an additional language detection step in the original methodology, which is now the only such step. The intuition is that any invalid C/C++ code detected by Guesslang would be rejected by Cppcheck.

A second challenge for re-implementation was that the authors did not specify the exact Cppcheck version used in their study and did not respond to multiple inquiries. Ultimately, we resorted to brute force by testing 15 different versions of Cppcheck on the provided $Answer_w$ list against the reported

Table 3: Side-by-side summary of the main claims in Zhang et al.'s paper [84] for **RQ1** and **RQ2** and claims based on the results of our replication using *SOTorrent22*.

Original Results Based on SOTorrent18	Replication Results Based on SOTorrent22
RQ1 *What are the types of code weaknesses that are detected in C/C++ code snippets on Stack Overflow?*	
The authors found 36% (i.e., 32 out of 89) of all the C/C++ CWE types in C/C++ code snippets on Stack Overflow.	We found 37% (i.e., 33 out of 89) of all the C/C++ CWE types in C/C++ code snippets on *SO*. CWE-476 is a newly introduced type for snippets after December 2018 and has 1,159 instances.
The authors identified 12,998 CWE instances within the latest versions of the 7,481 answers.	We identified 7,679 CWE instances within the latest versions of the 5,721 answers.
The authors found CWE-758 to be the sixth most prevalent CWE type in C/C++ code snippets with 482 (3.7%) instances.	We found CWE-758 to be the second most prevalent CWE type in C/C++ code snippets with 10,911 instances.
The authors found 10,533 CWE instances in the TOP-6 most prevalent CWE types affecting C/C++ code snippets on *SO*.	While we found 42,984 CWE instances in the TOP-6 most prevalent CWE types affecting C/C++ code snippets on *SO*.
RQ2 *How does code with weaknesses evolve through revisions?*	
As the number of revisions increases from one to ≥ 3, the proportion of improved $Code_w$ increases from 30.1% to 41.8%.	As the number of revisions increases from one to ≥ 3, the proportion of improved $Code_w$ increases from 3.1% to 7.4%.
In $Code_w$ with different rounds of revisions, a larger proportion of code snippets have reduced rather than increased the number of security weaknesses.	We observed a smaller proportion of code snippets whose associated security weaknesses reduced with different rounds of code revisions.
The authors found 92.6% (i.e., 10,884) of the 11,748 $Code_w$ had weaknesses introduced when their code snippets were initially created on Stack Overflow. They found 10,884 $Code_w$ introduced in the snippets' first version.	We found 93.1% (i.e., 28,155) of the 30,254 $Code_w$ had weaknesses introduced in their first version. However, we discovered significantly more $Code_w$ introduced when code snippets were initially created: 28,155.
69% (i.e., 8,103 out of 11,748) of the $Code_w$ has never been revised.	80.6% (i.e., 24,388 out of 30,254) of $Code_w$ has never been revised.

results in the paper. We found version v1.86 to be closest to the original results. This version detected $Answer_w = 15,724$, of which 11,142 (99.2% of 11,235) were also identified by the authors. We surmise the additional 4,489 answers result from erroneous filtering with Guesslang in the original approach.

We evaluated our re-implementation with the *SOTorrent18* data set that was also used by the authors, aiming to verify that our approach is within an acceptable error margin from the authors' results and allowing us to replicate their study with a newer data set faithfully. Table 6 in Appendix A compares with the original findings and shows that our approach resembles the original methodology closely enough.

6.1.3 Replication

We replicate their findings using *SOTorrent22*, released four years after the original dataset. We use Cppcheck v2.13 to show how the results would differ if the study were conceived years later. Table 8 in Appendix A shows results for different combinations of Cppcheck and SOTorrent versions.

Table 2 compares the original results with the replication based on *SOTorrent22* and Cppcheck v2.13. We found that the number of code snippets with *LoC* >= 5 increased between 2018 and 2022 (724,784 ↗ 938,643), a growth rate of 29.5%. Among these, the $Code_w$ also increased (11,748 ↗ 30,254), a growth rate of 157.5%.

Table 3 presents a side-by-side comparison of the authors' claims and our findings based on the newer dataset for the authors' **RQ1** and **RQ2**. Below, we summarize the main points.

Revisiting RQ1 Findings: We found that an additional CWE type (CWE-476) has appeared since December 2018, which is now the sixth most prevalent CWE type. Similarly, the authors identified 12,998 CWE instances in the latest versions of 7,481 answers, whereas we found 7,679 instances in 5,721 answers. Further, we noticed a shift in the ranking of CWE types: CWE-758 climbed 6th ↗ 2nd; CWE-401 dropped 2nd ↘ 3rd; CWE-775 fell 3rd ↘ 7th.

> We found that several of Zhang et al.'s [84] original conclusions regarding the types of code weaknesses are no longer valid. *SOTorrent22* contains proportionally more vulnerable snippets with different ratios for CWE types and the emergence of a new CWE type.

Revisiting RQ2 Findings: The authors noted that as the number of revisions to $Code_w$ increased from 1 to 3+, the proportion of *improved* $Code_w$ rose from 30.1% to 41.8% (see Table 7 in Appendix A). As a result, the authors concluded that a larger proportion of $Code_w$ has reduced rather than increased, indicating that revisions improve code security. In contrast, our results showed significantly lower improvement rates. As revisions increased from 1 to 3 or more, the proportion of improved $Code_w$ only rose from 3.1% to 7.4%.

> Our replication shows that if the authors had conducted their study 4 years later, they would have observed only a slight increase in the proportion of *improved* $Code_w$.

Revisiting RQ3 Findings: The authors found that *"the majority of the C/C++ Version$_w$ were contributed by a small number of users"* and that *"72.4 percent (i.e., 10,652) of Version$_w$ were posted by 36 percent (i.e., 2,292) of users."* In our replication, we found a shift where an even smaller number of users contributed $Version_w$, see Figure 7. We found that 72.4 percent (i.e., 35,034) of $Version_w$ were posted by 25 percent (i.e., 3,205) of users. In our data set, 36 percent

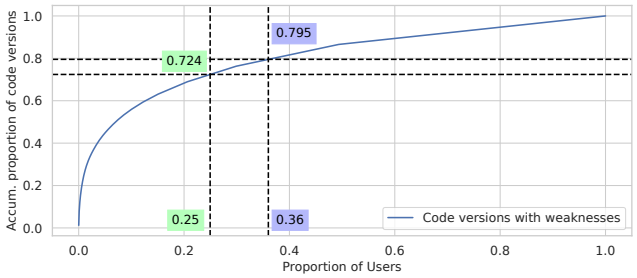


Figure 7: The accumulative proportion of $Version_w$ posted by the proportion of users. Annotations indicate the original (cf. Figure 9 in [84]) and replicated significant data points.

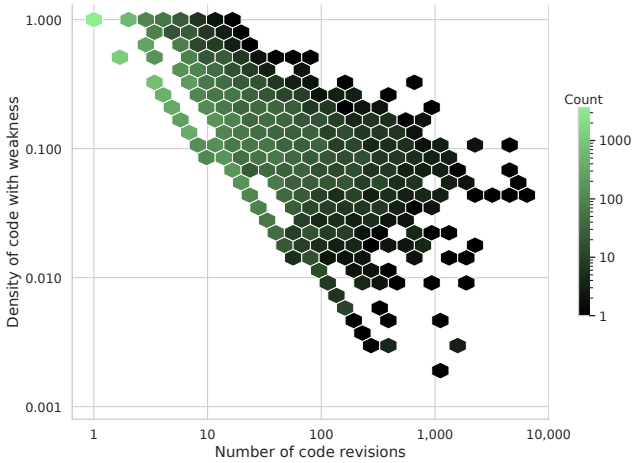


Figure 8: As the number of code revisions increases for a user, the density of contributed $Version_w$ by that user drops. Based on data from replication study. (cf. Figure 10 in [84])

(i.e., 4,625) of users contributed 79.5 percent (i.e., 38,481) of the $Version_w$. Moreover, Zhang et al. reported that *"64.0 percent (i.e., 4,070) of the users who contribute $Version_w$ have contributed only one $Version_w$."* We found that 86.2 percent (i.e., 11,077) of users contributed only one $Version_w$. Further, they reported that *"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."* In contrast, in our replication study, 17.0 percent (i.e., 12,845) of 75,779 users contributed code snippets with weaknesses.

Next, Zhang et al. explored the connection between user activity and code weaknesses. They found that "more active users are less likely to introduce $Code_w$." We depict the same connection in Figure 8, adopting the authors' plot style. The authors concluded that *"the weakness density of a user's code drops when the number of contributed code revisions by the user increases."* We explored the relation between the number of code revisions and the density of contributed $Version_w$ by users with statistical testing. A Pearson correlation analysis revealed a weak correlation, $r(4) = -0.190, p < 0.001$, suggesting that as the number of revisions increases, the density of code with weaknesses decreases slightly. A linear regression analysis was conducted to examine further the relationship between the number of revisions (independent variable) and the weakness density (dependent variable). The results indicated that the model was statistically significant $F(1, 12843) = 478.4$, $p < 0.001$, suggesting a significant inverse association of code revision count with weakness density. However, the model explained only a small proportion of the variance in weakness density $R^2 = 0.036$. These findings suggest that the weakness density is expected to decrease by 0.0005 for each additional revision. However, the low R^2 value indicates that the number of revisions explains only 3.6% of the variability in weakness density, suggesting that other factors may also play a significant role. Overall, the correlation and regression analyses support the conclusion that there is a statistically significant but weak inverse relationship between the number of revisions and weakness density. Further research is needed to explore additional variables and potential non-linear relationships that might better explain the variability in weakness density.

Additional details on the replication results for Zhang et al.'s RQ3 and a note about a possible multiplicity in counting CWE instances can be found in our extended version [45].

> We found that the fraction of users with vulnerable C/C++ snippets more than doubled compared to the original findings. Moreover, the number of users that contributed just one vulnerable snippet also increased. Further, the authors reported that users who contributed multiple vulnerable snippet versions repeatedly contributed the same CWE type. We found that these users contribute different types of CWE with the same likelihood.

6.2 Case Study 2: Discovering Insecure Code

Hong et al. [37] built DICOS to discover insecure code snippets by examining snippet revisions for changes in security-sensitive APIs, security-related keywords, and control flows. A code snippet is classified as insecure if at least two types of changes (see §3 in [37]) occurred between its initial and most recent version. The authors used tags to identify the programming language of code snippets (**R2**: ✔). Although the DICOS source code is available on GitHub [36], it contains several bugs that required fixing. Further, their dataset of labeled snippets is not available, even on request (**R1**: ○).

The authors followed a data-driven approach (**R4**: ✘) to answer: **RQ1** *Are older posts more likely to provide insecure code snippets?* **RQ2** *Are accepted answer posts more secure than non-accepted posts?* **RQ3** *What types of insecure code snippets were discovered?* **RQ4** *What is the status of reusing insecure code snippets in popular open-source software?*

We replicate the study for **RQ2** and **RQ3** to determine if their results still hold today. We excluded **RQ1** because code

evolution does not affect its findings, i.e., code evolution cannot retroactively *add* old posts, only evolve them. However, we used the results of **RQ1** to evaluate the released DICOS tool, which we will discuss later in this section. We excluded **RQ4** because it deals with code reuse from Stack Overflow in open-source projects on GitHub (**R3**: ✪, **S**). Though RQ4 is an interesting question for replication, our focus is on code evolution within Stack Overflow.

6.2.1 Original Methodology

The authors reported 93% precision, 94% recall, and 90% accuracy in discovering insecure **C/C++** code snippets with DICOS while for **Android** code snippets, it has 86% precision, 89% recall, and 86% accuracy. The evaluation was done using the *SOTorrent20* dataset, released in 12/2020, from which they extracted 668,520 posts containing 1,514,547 code snippets. For replication, we need to re-evaluate DICOS's precision, recall, and accuracy using the more recent SOTorrent22 dataset. The authors evaluated the accuracy of DICOS against the results by Fischer et al. [25] and Verdi et al. [78]. However, this methodology creates a barrier to replication: the authors only used the datasets provided by Fischer et al. and Verdi et al. rather than directly running DICOS and these tools on the same input. Specifically, they compared the insecure snippets identified by DICOS against the labeled code snippets from Fischer et al. and the insecure C++ snippets found by Verdi et al. To replicate this experiment, we would need to follow the same approach, using the labeled examples from related work to compare with DICOS's findings on a newer version of SOTorrent. Unfortunately, since these tools are unavailable and the labeled data from Fischer et al. and Verdi et al. only represent the old SOTorrent data set, this evaluation approach is infeasible when using a newer SOTorrent version.

6.2.2 Implementation

We used the DICOS code base to replicate the authors' findings on a newer dataset. We discovered several bugs, which we fixed for future use with DICOS, and noticed that some methods were only partially implemented. For example, Joern [3] was not fully used for control flow analysis, and their Jaccard index-based pairing technique produced many false positives. We plan to release a refactored DICOS version with the option to use clone detection as a pairing technique.

We evaluated the authors' DICOS implementation using the same dataset version to verify that we have a reproducible methodology as the basis for replication. However, we could not reproduce their reported Stack Overflow post counts using the same SQL queries [35]. While they reported 987,367 C/C++ and 970,916 Android posts, we found 867,962 C/C++ and 986,900 Android posts. Additionally, we identified 26,550 insecure posts, 14,092 more than the 12,458 reported. Unfortunately, we received no response from the authors for clarification. We did replicate their ***RQ1*** findings, observing a similar trend but with different yearly secure/insecure post counts. Details are in Appendix B. Although we could not reproduce the exact insecure post numbers, we replicated their study on a newer *SOTorrent* dataset version, using their code.

6.2.3 Replication

We used *SOTorrent22*, which was released *two* years after the *SOTorrent20* dataset used by the authors. We followed the authors' approach to collect posts from *SOTorrent22*. The authors extracted 1,958,283 posts (987,367 C/C++ and 970,916 Android), whereas we extracted 2,858,003 posts (1,489,148 C/C++ and 1,368,855 Android posts). This indicates a 51% increase in C/C++ posts and a 41% increase in Android posts since December 2020. The authors filtered all single-version posts, resulting in 668,520 (34% of 1,958,283) multi-version posts. After filtering all single-version posts from the extracted *SOTorrent22* posts, we obtained 1,046,052 posts (36.6% of 2,858,003) with at least two versions.

Finally, the authors applied DICOS on their dataset of 668,520 posts and discovered 12,458 (1.9%) insecure posts (8,941 C/C++ and 3,517 Android). Of these, 788 (6.3%) insecure posts contained all three types of changes while the remaining 11,670 insecure posts contained two types of changes. Using the same approach on our dataset of 1,046,052 posts, we discovered 30,359 (2.9%) insecure posts, i.e., an **increase of 52%**. Among these, 4,887 (16.1%) insecure posts contained all three features, i.e., an **increase of 155%**, while the remaining 25,472 posts contained two features.

***Accuracy of* DICOS**: To compute the accuracy, the authors manually verified a subset of the 12,458 discovered insecure posts using the following groups: **G1.** All posts with three changes **G2.** Top 200 posts with two changes **G3.** Randomly selected 100 posts with two changes **G4.** Top 200 posts with only one change **G5.** Top 100 posts without changes

Groups **G1–G3** were used to measure true and false positive rates for detecting insecure posts. Groups **G4** and **G5** were used to measure the true and false negative rates. Two researchers manually verified the posts for each group, recording the positive and negative rates for C/C++ and Android. We replicated this approach to calculate the precision, recall, and accuracy of DICOS using *SOTorrent22*. We adhered to the authors' original method, verifying 788 insecure posts for **G1** by randomly selecting 788 posts for manual verification by two researchers. For **G3**, we faced the challenge that the authors conducted a one-time random selection of 100 posts with two features. In contrast, we performed three random selections of 100 posts and averaged the results.

Table 4 compares the original accuracy of DICOS reported by the authors with our findings using a newer dataset. We found that DICOS had an 11% precision (compared to the authors' 91%), 32% accuracy (versus 89%), and an 87% recall (versus 93%). These results indicate that the performance of

Table 4: Comparison of precision, recall, and accuracy measured by authors (Table 5 in [37]) vs. our replication (SOTorrent22).

	Original accuracy measurement results based on *SOTorrent20*					Replicated accuracy measurement results based on *SOTorrent22*				
ID	**#Posts**	**#TP**	**#FP**	**#TN**	**#FN**	**#Posts**	**#TP**	**#FP**	**#TN**	**#FN**
G1	788	757	31	N/A	N/A	788	95	693	N/A	N/A
G2	400	346	54	N/A	N/A	400	33	367	N/A	N/A
G3	200	162	38	N/A	N/A	600	66	534	N/A	N/A
G4	400	N/A	N/A	318	82	400	N/A	N/A	379	17
G5	200	N/A	N/A	185	15	200	N/A	N/A	188	12
Total	1,988	1,265	123	503	97	2,388	194	1,594	567	29
Precision					**0.91**					**0.11**
Recall					**0.93**					**0.87**
Accuracy					**0.89**					**0.32**

DICOS has significantly decreased due to the code evolution on Stack Overflow. The replicated accuracy measurements for C/C++ and Android are in Appendix B.

> We found that code evolution has adversely affected the precision and accuracy of DICOS's approach to detecting vulnerable snippets. Considering the stable recall, DICOS on newer Stack Overflow versions is better suited for detecting *secure* snippets.

Revisiting RQ2 Findings: The authors investigated the relationship between the security weaknesses of code snippets in accepted and non-accepted answers and found no difference between the ratios of insecure posts for accepted (1.67%) and non-accepted (1.99%) answers. Using the newer version of the SOTorrent dataset, we observed a higher ratio of insecure posts between accepted (7.72%) and non-accepted (6.61%) answers. Figure 10 in Appendix B compares the original and replication results. A two-sample z-test for proportions of the insecure to secure posts ratio between the original and our replication results shows a significant difference ($Z = -126.888$, $p < 0.001$).

Revisiting RQ3 Findings: The authors manually categorized the 788 insecure posts containing all three types of changes by their weakness types. Figure 11 in Appendix B compares the original and replication results. The authors reported eight types of insecure code snippets, with undefined behavior (42%) being the most common. Our findings indicate that memory leaks (39.3%) are now the most prevalent security weakness in C/C++ code snippets. Furthermore, while the authors identified null-terminated strings as the second most common issue, our analysis found undefined behavior, out-of-bound errors, and others instead.

> For the newer data set version, the types of weaknesses and their frequencies have shifted.

6.3 Case Study 3: Snakes in Paradies

Rahman et al. [62] investigated Python code snippets with the aim of characterizing the prevalence of insecure Python-related coding practices. The authors used the SOTorrent18 dataset, released in 09/2018, to empirically answer the following research questions: **RQ1**: *How frequently do insecure coding practices appear in Python-related Stack Overflow answers?* **RQ2**: *How does user reputation relate to the frequency of insecure Python-related coding practices?* **RQ3**: *What are the characteristics of Python-related questions that include answers with insecure code practices?*

The authors did not investigate the reuse of Python snippets from Stack Overflow (**R3**: ✘) and followed a purely data-driven approach to answer their research questions (**R4**: ✘).

6.3.1 Original Methodology

The authors focused on Python code snippets in answer posts belonging to question posts that were viewed more than once and with a score >0 and that were attributed inside Python source files on GitHub. Attribution of question posts in Python projects was used to determine the language of snippets (**R2**: ✔). This resulted in 10,861 questions with 44,966 answers, from which they extracted 529,054 code snippets forming their final dataset. To detect insecure coding patterns, the authors used string matching to determine if a standard library or third-party Python API, known to be used in an insecure way, is found in a code snippet. The authors grouped the insecure Python APIs they considered in their study into six categories. Table II in the original paper describes them and their corresponding insecure coding patterns.

6.3.2 Implementation

The authors published their code artifacts but not their data artifacts (**R1**: ◑). We used the published code [2] for replication and collected our dataset from SOTorrent22. The data collection was straightforward since the authors extensively described their data collection approach. We tested our implementation with the data set the authors used (SOTorrent18) and came to the same numbers as in the original paper.

6.3.3 Replication

We replicated their findings using *SOTorrent22*, released four years after the original dataset version. After applying the au-

Table 5: Comparison of the results by Hong et al. [37] and our replication using *SOTorrent22*.

DICOS with SOTorrent20	DICOS with SOTorrent22
Data Collection	
The authors extracted 987,367 C/C++ posts and 970,916 Android posts, totaling 1,958,283 answer posts.	We extracted 1,460,627 C/C++ posts and 1,339,692 Android posts, totaling 2,800,319 answer posts. This is a 43% increase in the number of C/C++ and Android answer posts created on Stack Overflow after December 2020.
After filtering out single-version posts, the authors collected 668,520 multi-version answers, which they used to evaluate DICOS.	We collected 1,046,052 multi-version answers after filtering out single-version posts.
The authors found 12,458 insecure posts; 8,941 insecure C/C++ posts, and 3,517 insecure Android posts.	In contrast, we found 30,359 insecure posts; 22,167 insecure C/C++ posts and 8,192 insecure Android posts.
DICOS has 91% precision, 93% recall, and 89% accuracy.	We observed 11% precision, 87% recall and 32% accuracy.
RQ2 *Are accepted answer posts more secure than non-accepted posts?*	
The ratio of insecure posts was almost the same between accepted (1.67%) and non-accepted (1.99%) posts.	The ratio of insecure posts is very similar between accepted (7.72%) and non-accepted (6.61%) posts, and the overall ratio of insecure posts increased.
RQ3 *What types of insecure code snippets were discovered?*	
The most prevalent type of insecure code snippets was *undefined behavior*, accounting for 42% of the total;	The most prevalent type of insecure code snippets was *memory leak*, accounting for 39.25% of the total;
Authors observed *Null-terminated string issue* as the second most prevalent security weakness.	In contrast, we found three security weaknesses as the second most prevalent weaknesses: *Undefined behavior*, *Out-of-bounds error*, and *Others*.

thors' filtering criteria, we obtained 12,095 questions containing 72,202 answers, of which 10,140 were accepted answers. This means the number of code snippets matching the authors' filtering criteria has dropped since 2018: 529,054 ↘ 239,575.

Revisiting RQ1 Findings: Our findings regarding the number of questions with at least one insecure answer differ significantly: 18.1% (out of 10,861) dropped to 4.9% (out of 12,095). Similarly, the percentage of accepted answers containing at least one insecure snippet also decreased: 9.8% (out of 7,444) ↘ 2.2% (out of 10,139). Although the vulnerability rankings from the original study remain unchanged, we observed a shift in the number of affected snippets: *code injection* increased (2,319 ↗ 5,734), while *insecure cipher* (564 ↘ 356), *insecure connection* (624 ↘ 276), and *data serialization* (153 ↘ 140) all dropped. Like the original study, no snippets were impacted by XSS vulnerabilities.

> We found fewer insecure and accepted answers than the original, with percentages dropping significantly. Vulnerability rankings remained consistent, but there were notable shifts in affected snippets, including an increase in *code injection* cases and decreases in others. No XSS vulnerabilities were found, as in the original study.

Revisiting RQ2 Findings: The authors answered this question by comparing the *normalized* reputation score of users that contributed at least one insecure code snippet with those that contributed answers with no insecure code snippets. Normalization was required to reduce bias since long-time Stack Overflow users tend to have higher reputations. Using Mann-Whitney U and Cliff's Delta, the authors found no significant difference between answer providers with high and low reputations, suggesting that both are equally likely to introduce insecure code snippets. We also found no significant difference between the two user groups. However, we observed different $p-$value (0.9 ↗ 6.2) and Cliff's delta (0.01 ↗ 0.03) values.

Revisiting RQ3 Findings: The authors employed Latent Dirichlet Allocation topic modeling to group questions with an insecure answer. They found that answers to questions about **web**, **string**, and **RNG** topics contain at least one insecure code snippet. In contrast, we found that answers to questions related to **web** topics are no longer associated with insecure code snippets but questions about **string** and **RNG** topics still have answers containing insecure code snippets.

> We found that user reputation *still* does not influence the likelihood of posting insecure code snippets. However, we discovered that the association between *web* topics and insecure code snippets is no longer given.

6.4 Case Study 4: Mining Rule Violations

Campos et al. [22] investigated JavaScript code snippets on Stack Overflow to answer: **RQ1**: *How commonplace are rule violations in JavaScript code snippets?* **RQ2**: *What are the most common rules violated in JavaScript code snippets?* **RQ3**: *Are JavaScript code snippets flagged with possible errors being reused in GitHub projects?* We replicated the study for **RQ1** and **RQ2**. We omitted RQ3 because it concerns code reuse from Stack Overflow while we focus on its evolution.

6.4.1 Original Methodology

The authors used a data-driven approach (**R4**: ✘) and focused on accepted answers with JavaScript tags (**R2**: ✔) in the SOTorrent18 dataset. The latest version is selected if a code snippet has multiple versions, resulting in 336,643 code snippets. Using the ESLint analysis tool, the authors found that **all code snippets** contain rule violations, with stylistic issues being the most prevalent violation, accounting for 82.9%.

6.4.2 Implementation

The authors made their source code and data publicly available (**R1**: ✪), enabling us to replicate their findings on a newer dataset. However, we found a discrepancy between their paper's data collection method and the released dataset. Although the paper explained discarding snippets with fewer than 10 LoC, their dataset includes **42,158** snippets with nine LoC. Since the script for data collection was not shared, reproducing the exact number of code snippets from SOTorrent18 using the original approach was impossible. Similarly, when we attempted to reproduce the author's findings for their **RQ1** using their artifacts, we could not come to their conclusion that *no code snippet in their dataset was free of violations*. Instead, we found 153,159 (45.5% of 336,643) snippets containing only parse errors but no rule violations. We contacted the authors for clarification and learned they chose a minimum LoC of nine and included parse errors as violations. In our replication study, we followed their clarification.

6.4.3 Replication

We replicated **RQ1** and **RQ2** using *SOTorrent22*, released four years after the dataset version used in their study.

Revisiting RQ1 Findings: The original study reported that no JavaScript code snippet was free of violations, but our replication found nine snippets without any violations. However, the number of violations in JavaScript code snippets increased from 5,587,357 to 7,385,044, with the average violations per snippet rising from 11.94 to 28.8.

> The observation that no JavaScript code snippet on Stack Overflow is violation-free no longer holds, and we found more violations in snippets than originally reported.

Revisiting RQ2 Findings: The original study grouped violations into six categories and reported the top three most common rule violations for each category:

Stylistic Issues: Violations increased by 28.4% (4,632,348 ↗ 5,946,283), with the three most common violations in this category being: (1) *semi*: 1,477,808 ↗ 1,990,461, (2) *quotes*: 700,770 ↗ 1,072,468, and (3) *no-trailing-spaces*: 374,012 to 473,294. The authors removed 3,461,739 violations related to indentation rule violations, reasoning that multiple snippets were merged into a single file. We followed the same approach and removed 4,910,529 indentation violations.

Variable: Violations increased by 28.7% (787,824 ↗ 1,013,612), with the top issues being: (1) *no-undef*: 719,679 ↗ 913,103, (2) *no-unused-vars*: 67,816 ↗ 100,124, and (3) *no-undef-init*: 150 ↗ 189.

Best Practices (BP): Violations increased by 216.5% (57,578 ↗ 182,232), with the most common being: (1) *eqeqeq*: 53,321 ↗ 66,664, (2) *no-multi-spaces*: 54,768 ↗ 50,352, (3) *no-redeclare*: 18624, and (4) *curly*: 14,989 ↗ 18,292. *eqeqeq* is now the most common, while *curly* dropped to fourth place.

Possible Errors (PE): Violations increased by 39% (6,303 ↗ 8,762), with the most common being: (1) *no-irregular-whitespace*: 2,037 ↗ 2,196, (2) *no-cond-assign*: 910 ↗ 1,196, and (3) *no-dupe-keys*: 2070, which replaced *no-unreachable* as the third most common violation.

Node.js/Common.js: Violations increased by 27.5% (3,304 ↗ 4,211), with the most common violations being: (1) *handle-callback-err*: 2,855 ↗ 3,650, (2) *no-path-concat*: 444 ↗ 555, and (3) *no-new-require*: 5 ↗ 6.

ECMAscript 6: Violations increased by 139.8% (from 548 to 1,314), with the most common being: (1) *template-curly-spacing*: 164 ↗ 516, (2) *no-useless-constructor*: 154 ↗ 238, and (3) *no-const-assign*: 129, replacing *no-this-before-super* as the third most common violation.

Parse Errors: We found 267,795 code snippets with *parse errors* but no violations. While the original study included parse errors in the total violation count, we introduced a seventh category to separately list snippets with only parse errors.

> Due to Stack Overflow's evolution, the number of violations in each category has risen. Further, the ranking of violations within certain categories has changed.

7 Limitations and Challenges

We briefly discuss our study's limitations and the basic challenges of studying the security of Stack Overflow snippets.

Quality of original studies. This study replicates prior research using the original methods on a newer dataset. Consequently, any flaws or biases in the original studies are also reflected in our replication.

Generalizability of results. Our study specifically targeted research investigating the security properties of Stack Overflow code snippets. Thus, our findings may not apply to Stack Overflow-based studies that explore other aspects, such as user behavior, limiting the generalizability of our conclusions.

Dependence on available artifacts. The original studies' quality and availability of the artifacts posed another limitation. If a published artifact inaccurately implemented a study's methodology, these inaccuracies carried over into our replication. Despite our best efforts to reuse the same source artifacts, any errors or bugs in the original implementation may have impacted our results. Therefore, the reliability of our replication study is inherently linked to the accuracy and completeness of the original research artifacts.

However, the biggest challenge in our study was the lack of artifacts from prior research (see Table 1), which forced us to re-implement the original methodology in various case studies. This was not always possible (e.g., training a machine learning-based classifier without access to the original data) and can be error-prone (e.g., replacing deprecated toolchains).

Language detection. A general challenge for studies on Stack Overflow is determining the programming language of code snippets. Related work relied on post tags or the Guesslang tool (see **R2** in Table 1). We evaluated these approaches briefly and found neither is reliable (see [45] for details). In comparison, ChatGPT outperformed either tool, but it is neither scalable in data set size nor economically reasonable for our replication studies to create a ground truth of snippet languages. Thus, future work could create a fine-tuned LLM to replace Guesslang for the language detection task.

Security classification. Several studies rely on a security classification of code snippets by either building/using dedicated tools for specific languages (**D2** in Table 1) or, in the absence of a generic code security classifier, leveraging the context of snippets (i.e., comments and commit messages; **D4** in Table 1). We used the tool by Jallow et al. [43] from the latter category. However, this tool has a potentially high false positive rate due to its keyword-based detection. Future work could investigate a more reliable context-based tool, e.g., taking inspiration from other NLP-based solutions [7, 33].

User studies. Several studies involved user studies [6, 11, 17, 26, 54, 55, 61, 63, 65] (**R4**, Table 1) and it is a limitation of this work that we did not replicate any human-centered research. Some of these prior works conducted developer surveys [11, 17, 26, 61] and are more amenable to replication. In contrast, other works involved developer studies to solve programming challenges with the help of StackOverflow [6] or to evaluate their tools [54, 58, 65]. These studies require a carefully crafted replication to isolate differences between participant groups from the impact of Stack Overflow evolution.

8 Related Works

8.1 Meta-research

Meta-research is an important tool for evaluating and improving research practices. Ioannidis [40] investigated prior research findings and found that most published research turned out to be false. One factor contributing to this crisis is the lack of code and data to verify research claims. Ioannidis et al. [41] introduced a framework that serves as a benchmark to holistically evaluate and improve research practices to make results more reliable. They identified methods, reporting, reproducibility, evaluation, and incentives as areas of interest for research practices. Demir et al. [20] examined the reproducibility and replicability of web measurement studies. They discovered that many studies lacked proper documentation of their experimental setups, which is essential for accurately reproducing and replicating results. In particular, they found that even slight variations in experimental setup could lead to significant differences in results. The authors recommended proper documentation and adopting standardized practices to make web measurement findings more reliable. Weber et al. [80] conducted a study of benchmarking studies to develop guidelines to help computational scientists conduct better benchmarking studies. While their guidelines are for a different target group than ours, some of them translate to our work. For instance, the guideline on selecting datasets and reproducible research best practices can be transferred, as it shows that security studies focused on Stack Exchange datasets should be measured using different dataset versions and that the code and data for studies should be made publicly available to facilitate replication studies.

8.2 Dataset Evolution

Ceroni et al. [15] investigated Wikipedia's dynamic nature to examine how its content changes over time, specifically looking at page edits. Their research highlights the significance of understanding content evolution to evaluate the quality and accuracy of information on Wikipedia. In a follow-up, Tran et al. [75] leverage the evolution of data on Wikipedia to analyze the history of user edits to extract and represent complex events. Fetterly et al. [23] provides a detailed investigation of how web pages evolve over time to understand how that evolution might impact crawling and indexing processes for the web. Their study shed light on the importance of understanding the evolution of web pages for improving web crawling and indexing processes. Jallow et al. [43] is closely related to us. They studied the impact of evolving Stack Overflow code snippets on software developers. In contrast, we focus on how code evolution might impact prior research findings.

Conducting time-series analysis to better understand data that can change over time is explained in the textbooks of several other research disciplines, such as economics [32], environmental science [81], medicine [69], finance [76], social sciences [14], or engineering [52].

9 Conclusions for Future Studies

Time-Series Analysis. Our systematization showed that security-focused Stack Overflow studies relied on specific dataset versions and non-stationary aspects. Among the 42 relevant papers we identified, only Ragkhitwetsagul et al. [61] discussed the potential impact of evolution on their results. Four works [37, 56, 68, 84] use code evolution in their methodology but do not discuss its consequences for their findings. With our six replication studies, we revealed that the findings

of four studies differ for newer dataset versions.[1] This does *not* mean that the results of prior research are wrong—but missing context. Researchers are advised to provide additional context around their results by treating the Stack Overflow dataset as the time-series data it is. Considering the temporal dimension of posts and conducting trend analysis can help better understand such results, e.g., whether issues are recurring or stable, and can address temporal dynamics, such as differentiating short-lived trends from long-term changes. The Stack Overflow dataset is particularly suited to this practice since the versioning of posts allows analysis of the platform's prior states. While it is unrealistic to predict future changes with certainty, considering multiple dataset versions and their aspects' stationarity provides insights into how changes on Stack Overflow might affect the findings. For example, we found that CWE types of C/C++ code snippets shifted over time (see §6.1) while crypto API misuse in Java snippets seemed stable or even increasing (see [45]). Trend analysis is common in other research disciplines, where drawing conclusions from a single observation would be insufficient, if not misleading. This is particularly true in fields such as economics [32] (studying variables like GDP, inflation, or stock prices), environmental science [81] (analyzing climate data or temperature changes), or social sciences [14] (examining voting patterns, crime rates, or demographic shifts). We suggest that our discipline adopt such a longitudinal analysis methodology when studying data that changes over time.

Open Science Best Practices. Although not a goal at the outset of our work, we want to re-iterate some best practices for open science and reproducibility that we found disregarded during our replication studies. We found that many artifacts were not or only partially available (on request) or non-functional. Making code and data available on request is often insufficient [80], and experience shows that paper authors can often ignore such requests. Thus, our work underlines the need to establish better efforts for open science, and we welcome the recent steps to introduce artifact evaluation committees and require artifacts for accepted papers. Moreover, we also found that the information provided in the papers was often insufficient to allow others to re-implement the methods.Both the ACM SIGPLAN Empirical Evaluation Guidelines [4] and van der Kouwe et al.'s benchmarking flaws [77] remark that lack of such information leads to a lack of reproducibility and can hinder scientific progress. Thus, we urge researchers to report the software versions used in their methods. Containerization tools, like Docker, could encapsulate software environments and preserve package versions and dependencies, easing replication efforts and artifact releases.

[1]The two studies with stable results are in our extended version [45].

Ethics Considerations and Compliance with the Open Science Policy

Open Science and Availability

The artifact for this paper is publicly accessible and permanently hosted on Zenodo [44].

Ethics Considerations

Reproducibility and Reliability. We tried to consider the implications of our findings for the broader field. Where our replication studies produced different results from the original study, we tried to discuss the possible reasons for these differences and their impact on the original findings. We aim to identify the potential causes for diverging results in a systematic way before our replication study in §3 and §4.

Originality and Plagiarism. We acknowledge the original studies we replicated. We avoid plagiarism by properly crediting the original authors and citing their work appropriately.

Respect for Original Research. We tried to be respectful of the original researchers' work. Critiques or discussions regarding the original study are phrased constructively and based on the data and findings.

Consent for the Use of Original Data. All data and code artifacts of the prior studies were either publicly available (on request) or re-implemented based on the published results. We credited the original researchers appropriately if their artifacts were reused, and our systematization in Section 3 highlights which artifacts were available.

Communication with Original Researchers. We engaged with the original researchers, especially when we could not reproduce the original results and when artifacts were unavailable or seemingly flawed. This was intended to help ensure that any discrepancies are thoroughly understood and addressed in a collaborative manner.

Responsible Disclosure and Report Findings. Some of the replicated studies [25, 84] aim at discovering vulnerable code snippets. Replicating their methodology on a newer Stack Overflow data set can result in discovering additional snippets with weaknesses. While discovered vulnerabilities in other code bases (e.g., production or open-source software) should be responsibly disclosed to vendors/developers, there are no guidelines about disclosure for publicly posted code snippets. While the technical option exists to add a comment[2] on each post with a vulnerable snippet, we found this to be an ethical dilemma. First, the vulnerability does not affect

[2]`https://api.stackexchange.com/docs/create-comment`

Stack Overflow but only developers that copy such snippets into their code bases. Thus, the disclosure might not reach the affected parties. Second, the employed methodologies only discover weaknesses, but without a concrete explanation of the vulnerability and the corresponding fix, a comment would not benefit the community. Third, without careful consideration of the context of the snippet, commenting on weaknesses can easily create noise and even be considered offensive/useless (e.g., if the original question asked about an explanation of a vulnerability, i.e., the vulnerability was posted on purpose). Without being able to address the second and third concerns at scale, such auto-commenting of weaknesses would also violate the Stack Overflow code of conduct.[3] We further consulted the Stack Exchange Meta discourse[4] about how vulnerabilities in code snippets should be reported. The meta-discussions mirror, to some extent, our concerns. For example, that mass commenting will create noise[5] and that such discoveries should be accompanied by code fixes[6]. The community also discussed other measures, such as introducing a "security" or "caution" flag to posts with vulnerabilities or outdated security measures.[7] However, the community also agrees on the sentiment that while code snippets could be copied&pasted, Stack Overflow is a teaching platform for developers to learn how to solve their specific problems and that authored code should be owned, hence, not just pasted and instead, the solutions on the platform should be scrutinized and lead to further education and ultimately custom solutions.[8] Since it is currently impossible to make comments/edits of the required quality, we did not create fixes for newly discovered posts with weaknesses for the above-stated reasons. We note that neither Zhang et al. [84] nor Fischer et al. [25] mentioned any attempted disclosure on Stack Overflow—Fischer et al., however, in follow-up work [27], built a browser plugin to warn Stack Overflow users of insecure snippets.

References

[1] "Artifact for Paper: Measuring the Effects of Stack Overflow Code Snippet Evolution." [Online]. Available: https://osf.io/s2vgm/?view_only=785ada7b1efd4ac6aaf5a77cc5123076

[2] "Artifact for Paper: Snakes in Paradies." [Online]. Available: https://figshare.com/s/588b0d450310c05d25ab?file=14644805

[3] Joern: The bug hunter's workbench. [Online]. Available: https://joern.io/

[4] "SIGPLAN Empirical Evaluation Guidelines." [Online]. Available: https://www.sigplan.org/Resources/EmpiricalEvaluation/

[5] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from stackoverflow: An exploratory study on android apps," *Information and Software Technology*, vol. 88, 2017.

[6] 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. 37th IEEE Symposium on Security and Privacy (SP '16)*, 2016.

[7] O. Akgul, S. T. Peddinti, N. Taft, M. L. Mazurek, H. Harkous, A. Srivastava, and B. Seguin, "A decade of Privacy-Relevant android app reviews: Large scale trends," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[8] M. Alhanahnah and Q. Yan, "Towards best secure coding practice for implementing ssl/tls," in *IEEE Conference on Computer Communications Workshops*, 2018.

[9] L. Almeida, M. Gonzaga, J. F. Santos, and R. Abreu, "Rexstepper: a reference debugger for javascript regular expressions," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings*, 2023.

[10] M. Bagherzadeh, N. Fireman, A. Shawesh, and R. Khatchadourian, "Actor concurrency bugs: a comprehensive study on symptoms, root causes, api usages, and differences," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, Nov. 2020.

[11] W. Bai, O. Akgul, and M. L. Mazurek, "A qualitative investigation of insecure code propagation from online forums," in *2019 IEEE Cybersecurity Development (SecDev)*, 2019, pp. 34–48.

[12] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1259–1295, Jun 2019.

[13] S. Baltes, C. Treude, and S. Diehl, "Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets," in *Proc. 16th International Conference on Mining Software Repositories (MSR 2019)*, 2019.

[14] J. M. Box-Steffensmeier, J. R. Freeman, M. P. Hitt, and J. C. W. Pevehouse, *Time Series Analysis for the Social Sciences*, ser. Analytical Methods for Social Research. Cambridge University Press, 2014.

[15] A. Ceroni, M. Georgescu, U. Gadiraju, K. D. Naini, and M. Fisichella, "Information evolution in wikipedia," in *Proceedings of The International Symposium on Open Collaboration*, ser. OpenSym '14. ACM, 2014.

[16] M. Chakraborty, "Does reusing pre-trained nlp model propagate bugs?" in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2021.

[17] F. Chen and S. Kim, "Crowd debugging," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. ACM, 2015.

[3] https://stackoverflow.com/help/privileges/comment

[4] https://meta.stackexchange.com/

[5] https://meta.stackexchange.com/questions/258328/right-approach-to-crawl-and-identify-bad-code

[6] https://meta.stackexchange.com/questions/9460/how-to-deal-with-questions-answers-with-a-security-vulnerability

[7] https://meta.stackexchange.com/questions/301592/keeping-answers-related-to-security-up-to-date, https://meta.stackexchange.com/questions/89469/what-to-do-with-questions-with-harmful-content/89474

[8] https://meta.stackexchange.com/questions/334811/stack-overflow-made-the-bbc-news-copycat-coders-create-vulnerable-apps

[18] L. Chen, S. Hou, Y. Ye, T. Bourlai, S. Xu, and L. Zhao, "itrustso: An intelligent system for automatic detection of insecure code snippets in stack overflow," in *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2019.

[19] M. Chen, F. Fischer, N. Meng, X. Wang, and J. Grossklags, "How reliable is the crowdsourced knowledge of security implementation?" in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019.

[20] N. Demir, M. Große-Kampmann, T. Urban, C. Wressnegger, T. Holz, and N. Pohlmann, "Reproducibility and replicability of web measurement studies," in *Proc. ACM Web Conference*, 2022.

[21] D. Evans, "Systematizing systematization of knowledge." [Online]. Available: https://oaklandsok.github.io/

[22] U. Ferreira Campos, G. Smethurst, J. P. Moraes, R. Bonifácio, and G. Pinto, "Mining rule violations in javascript code snippets," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019.

[23] D. Fetterly, M. Manasse, M. Najork, and J. Wiener, "A large-scale study of the evolution of web pages," in *12th International Conference on World Wide Web*. ACM, 2003.

[24] E. Firouzi, A. Sami, F. Khomh, and G. Uddin, "On the use of c# unsafe code context: An empirical study of stack overflow," in *Proc. 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2020.

[25] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *Proc. 38th IEEE Symposium on Security and Privacy (SP '17)*. IEEE Computer Society, 2017.

[26] F. Fischer, Y. Stachelscheid, and J. Grossklags, "The effect of google search on software security: Unobtrusive security interventions via content re-ranking," in *Proc. 28th ACM Conference on Computer and Communication Security (CCS)*, 2021.

[27] F. Fischer, H. Xiao, C.-Y. Kao, Y. Stachelscheid, B. Johnson, D. Razar, P. Fawkesley, N. Buckley, K. Böttinger, P. Muntean, and J. Grossklags, "Stack overflow considered helpful! deep learning security nudges towards stronger cryptography," in *Proc. 28th USENIX Security Symposium (SEC)*, 2019.

[28] W. Fuller, *Introduction to statistical time series*, ser. A Wiley publication in applied statistics. Wiley, 1976.

[29] Q. Gao, H. Zhang, J. Wang, Y. Xiong, L. Zhang, and H. Mei, "Fixing recurring crash bugs via analyzing q&a sites (t)," in *Proc. 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015.

[30] A. Ghanbari, D.-G. Thomas, M. A. Arshad, and H. Rajan, "Mutation-based fault localization of deep neural networks," in *Proc. 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 2024.

[31] Guesslang. (2021) Guesslang documentation. [Online]. Available: https://guesslang.readthedocs.io/en/latest/

[32] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.

[33] H. Harkous, S. T. Peddinti, R. Khandelwal, A. Srivastava, and N. Taft, "Hark: A deep learning system for navigating privacy feedback at scale," in *43rd IEEE Symposium on Security and Privacy (SP'22)*, 2022.

[34] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[35] H. Hong. (2022) Data collection sql script. [Online]. Available: https://github.com/hyunji-Hong/Dicos-public/blob/main/src/sql/collecting_allhisotyPost.sql

[36] ——. (2022) Dicos github repository. [Online]. Available: https://github.com/hyunji-Hong/Dicos-public

[37] H. Hong, S. Woo, and H. Lee, "Dicos: Discovering insecure code snippets from stack overflow posts by leveraging user discussions," in *Annual Computer Security Applications Conference (ACSAC)*. ACM, 2021.

[38] H. Imai and A. Kanaoka, "Time series analysis of copy-and-paste impact on android application security," in *13th Asia Joint Conference on Information Security (AsiaJCIS)*, 2018.

[39] S. E. Inc. (2024) Data explorer. [Online]. Available: https://data.stackexchange.com/stackoverflow/query/1858561/total-number-of-question-and-answer-posts

[40] J. P. A. Ioannidis, "Why most published research findings are false," *PLOS Medicine*, vol. 2, no. 8, p. null, 08 2005.

[41] J. P. A. Ioannidis, D. Fanelli, D. D. Dunne, and S. N. Goodman, "Meta-research: Evaluation and improvement of research methods and practices," *PLOS Biology*, vol. 13, no. 10, 2015.

[42] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, "Repairing deep neural networks: fix patterns and challenges," in *Proc. ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. ACM, 2020.

[43] A. Jallow, M. Schilling, M. Backes, and S. Bugiel, "Measuring the effects of stack overflow code snippet evolution on open-source software security," in *45th IEEE Symposium on Security and Privacy (SP'24)*. IEEE, 2024.

[44] A. Jallow and S. Bugiel, "Dataset and code for: Stack overflow meets replication: Security research amid evolving code snippets," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.14759484

[45] ——, "Stack overflow meets replication: Security research amid evolving code snippets (extended version)," 2025. [Online]. Available: https://arxiv.org/abs/2501.16948v2

[46] H. Y. Jhoo, S. Kim, W. Song, K. Park, D. Lee, and K. Yi, "A static analyzer for detecting tensor shape errors in deep neural network training code," in *Proc. ACM/IEEE 44th International Conference on Software Engineering (ICSE)*, 2022.

[47] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, 01 2007.

[48] D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" *Journal of Econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992.

[49] S. A. Licorish and T. Nishatharan, "Contextual profiling of stack overflow java code security vulnerabilities initial insights from a pilot study," in *21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE Computer Society, 2021.

[50] S. A. Licorish and M. Wagner, "Dissecting copy/delete/replace/swap mutations: insights from a gin case study," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '22. ACM, 2022.

[51] Y. Liu, Y. Yan, C. Sha, X. Peng, B. Chen, and C. Wang, "Deepanna: Deep learning based java annotation recommendation and misuse detection," in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2022.

[52] L. Ljung, *System identification: theory for the user*. Prentice-Hall, Inc., 1986.

[53] M. Madsen, O. Lhoták, and F. Tip, "A model for reasoning about javascript promises," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, Oct. 2017. [Online]. Available: https://doi.org/10.1145/3133910

[54] S. Mahajan, N. Abolhassani, and M. R. Prasad, "Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. ACM, 2020.

[55] S. Mahajan and M. R. Prasad, "Providing real-time assistance for repairing runtime exceptions using stack overflow posts," in *IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2022.

[56] S. S. Manes and O. Baysal, "Studying the change histories of stack overflow and github snippets," in *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021.

[57] N. Meng, S. Nagy, D. Yao, W. Zhuang, and G. Arango-Argoty, "Secure coding practices in java: Challenges and vulnerabilities," in *IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018.

[58] K. Mindermann and S. Wagner, "Fluid intelligence doesn't matter! effects of code examples on the usability of crypto apis," in *Proc. ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. ACM, 2020.

[59] M. Moradi Moghadam, M. Bagherzadeh, R. Khatchadourian, and H. Bagheri, "μakka: Mutation testing for actor concurrency in akka using real-world bugs," in *Proc. 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2023.

[60] R. Pan, "Does fixing bug increase robustness in deep learning?" in *Proc. ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings (ICSE)*, 2020.

[61] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, 2021.

[62] A. Rahman, E. Farhana, and N. Imtiaz, "Snakes in paradise? insecure python-related coding practices in stack overflow," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19. IEEE Press, 2019.

[63] M. S. Rahman and C. K. Roy, "An insight into the reusability of stack overflow code fragments in mobile applications," in *IEEE 16th International Workshop on Software Clones*, 2022.

[64] A. Reinhardt, T. Zhang, M. Mathur, and M. Kim, "Augmenting stack overflow with api usage patterns mined from github," in *Proc. 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2018.

[65] X. Ren, J. Sun, Z. Xing, X. Xia, and J. Sun, "Demystify official api usage directives with crowdsourced api misuse scenarios, erroneous code examples and patches," in *IEEE/ACM 42nd International Conference on Software Engineering*, 2020.

[66] H. Schmidt, M. van Aerssen, C. Leich, A. Benni, S. Al Ali, and J. Tanz, "Copypastavulguard – a browser extension to prevent copy and paste spreading of vulnerable source code in forum posts," in *Proc. 17th International Conference on Availability, Reliability and Security (ARES)*. ACM, 2022.

[67] Sebastian Baltes. (2022) Sotorrent post history extractor. [Online]. Available: https://github.com/sotorrent/posthistory-extractor

[68] M. Selvaraj and G. Uddin, "Does collaborative editing help mitigate security vulnerabilities in crowd-shared iot code examples?" in *Proc. 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2022.

[69] R. Shumway and D. Stoffer, *Time Series Analysis and Its Applications With R Examples*, 2011, vol. 9.

[70] A. Soni and S. Nadi, "Analyzing comment-induced updates on stack overflow," in *Proc. 16th International Conference on Mining Software Repositories (MSR '19)*. IEEE Press, 2019.

[71] Stack Exchange, Inc., "Stack Exchange Data Dump," accessed 2023-06-13. [Online]. Available: https://archive.org/details/stackexchange/

[72] Stack Exchange Meta. (2022) Academic papers using stack exchange data. [Online]. Available: https://meta.stackexchange.com/questions/134495/academic-papers-using-stack-exchange-data

[73] Stack Overflow. (2024) Revisions of answer post 14424800. [Online]. Available: https://stackoverflow.com/posts/14424800/revisions

[74] N. M. Synovic, M. Hyatt, R. Sethi, S. Thota, Shilpika, A. J. Miller, W. Jiang, E. S. Amobi, A. Pinderski, K. Läufer, N. J. Hayward, N. Klingensmith, J. C. Davis, and G. K. Thiruvathukal, "Snapshot metrics are not enough: Analyzing software repositories with longitudinal metrics," in *Proc. 37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2023.

[75] T. Tran, A. Ceroni, M. Georgescu, K. Djafari Naini, and M. Fisichella, "Wikipevent: Leveraging wikipedia edit history for event detection," in *International Conference on Web Information Systems Engineering*. Springer, 2014.

[76] R. Tsay, *Analysis of financial time series*, 2nd ed., ser. Wiley series in probability and statistics. Wiley-Interscience, 2005.

[77] E. van der Kouwe, G. Heiser, D. Andriesse, H. Bos, and C. Giuffrida, "Sok: Benchmarking flaws in systems security," in *European Symposium on Security and Privacy (EuroSP)*. IEEE Computer Society, 2019.

[78] 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," *arXiv:1910.01321*, 2019.

[79] H. Vocke, "We're switching to commonmark," https://meta.stackexchange.com/questions/348746/were-switching-to-commonmark, 2020.

[80] L. M. Weber, W. Saelens, R. Cannoodt, C. Soneson, A. Hapfelmeier, P. P. Gardner, A.-L. Boulesteix, Y. Saeys, and M. D. Robinson, "Essential guidelines for computational method benchmarking," *Genome biology*, vol. 20, 2019.

[81] D. S. Wilks, *Statistical methods in the atmospheric sciences*. Elsevier Academic Press, 2011.

[82] A. Yadavally, T. N. Nguyen, W. Wang, and S. Wang, "(partial) program dependence learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.

[83] Y. Ye, S. Hou, L. Chen, X. Li, L. Zhao, S. Xu, J. Wang, and Q. Xiong, "Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. ACM, 2018.

[84] H. Zhang, S. Wang, H. Li, T. Chen, and A. E. Hassan, "A study of c/c++ code weaknesses on stack overflow," *IEEE Transactions on Software Engineering*, vol. 48, no. 07, jul 2022.

[85] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online q&a forum reliable?: A study of api misuse on stack overflow," in *Proc. 40th International Conference on Software Engineering (ICSE'18)*, 2018.

[86] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proc. 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2018.

A Additional Details About Case Study 1

Table 6 presents a detailed, side-by-side comparison of the results from the original methodology by Zhang et al. [84] and our re-implementation, which utilized Cppcheck 1.86 for both language and security weakness detection (see Figure 6 for details). We identified 15,724 C/C++ answers containing security weaknesses. Of these, 11,142 answers were also flagged as containing weaknesses in the original study's $Answer_w = 11,235$. This indicates that our approach missed only 93 answers with weaknesses from the original study, demonstrating that our approach captures the same snippets as the original methodology. We surmise that the increase in detected snippets and snippet versions in our methodology is rooted in Guesslang's performance in the original methodology, i.e., many C/C++ snippets were misclassified as another language and hence not analyzed with Cppcheck by the authors. However, since the exact Cppcheck version used by the authors is unknown and we empirically estimated it to be version 1.86, we cannot exclude that our guessed version differs and performs differently from the authors' version.

Table 7 compares the proportion of $Code_w$ versus the number of code revisions that were detected with the original methodology by Zhang et al. [84] for SOTorrent18 and our re-implementation based on SOTorrent22.

Table 8 lists the vulnerable answers, snippets, and versions detected with different versions of Cppcheck on different versions of SOTorrent. A Shapiro-Wilk test confirmed the normality of the distribution of differences between paired observations for the SOTorrent versions for Cppcheck v1.86 ($0.832, p = 0.193$) and v2.13 ($0.852, p = 0.247$). A paired t-test indicates that the effect of upgrading the SOTorrent dataset is not consistent across Cppcheck versions ($t = -8.90, p < 0.05$). The change from SOTorrent18 to SOTorrent22 significantly impacted the results when using Cppcheck v2.13 compared to v1.86, implying that changes in the dataset can have a different impact depending on the Cppcheck version. Although using Cppcheck v1.86 on SOTorrent22 would better isolate the effect of code evolution, we decided to report the results of Cppcheck v2.13 in our replication study. We reason that if the authors had conducted their experiment later, they would have used the newer version.

B Additional Details About Case Study 2

We replicated the accuracy measurements for C/C++ and Android posts as presented in Table 3 and Table 4 of the original paper [37]. Our comparative findings for C/C++ posts and Android are in our extended version [45]. For C/C++ posts, we observed an accuracy of **11%** (authors reported 93%), a recall of 92% (vs. 94%), and a precision of **27%** (vs. 90%). Similarly, for Android posts, we found a significant drop in precision (12% vs. 86%) and accuracy (41% vs. 86%) while recall remained high (78% vs. 89%).

As stated in §6.2, we reproduced the authors' results for **RQ1** using the same dataset version. Our comparative findings are shown in Figure 9. The authors concluded that older posts are less likely to introduce insecure posts than newer ones. While we observed similar trends, we found different yearly numbers of secure/insecure posts from those reported by the authors. For instance, in 2008, the authors observed 63 insecure and 2,446 secure posts while we observed 82 insecure and 1,292 secure posts.

A two-sample z-test for the *overall* proportions of the insecure to secure posts ratio between the original and our replication results shows a significant difference ($Z = -121.962$, $p < 0.001$). To evaluate whether the proportion of insecure posts differed significantly between the originally reported results and our replication study, we conducted two-proportion z-tests for each year from 2008 to 2020. The Holm–Bonferroni correction [34] was applied to account for multiple comparisons across the 13 years. Our results indicate that for all years, the differences in the proportion of insecure posts between the two studies are statistically significant after the Holm–Bonferroni correction (adjusted α levels ranged

Table 6: Comparison of results by Zhang et al. (cf. Table 1 in [84]) and our evaluation using Cppcheck v1.86.

SOTorrent18 (Original)				SOTorrent18 (Evaluation)			
	Answer #	*Code Snippet #*	*Code Version #*		*Answer #*	*Code Snippet #*	*Code Version #*
SOTorrent	867,734	1,561,550	1,833,449	SOTorrent	867,734	1,561,550	1,833,449
LOC >= 5	527,932	724,784	919,947	LOC >= 5	527,932	724,784	919,947
Guesslang	490,778	646,716	826,520	Cppcheck *v*1.86	141,215	170,974	206,582
$Code_w$	**11,235**	**11,748**	**14,934**	$Code_w$	**15,724**	**16,533**	**20,664**

Table 7: The proportion of $Code_w$ versus the number of code revisions by Zhang et al. [84] and our replication study using SOTorrent22 and Cppcheck v2.13. (cf. Table 2 in [84])

	Original results based on SOTorrent18				Replication results based on SOTorrent22			
#revisions	*Snippets*	*Unchanged*	*Improved*	*Deteriorated*	*Snippets*	*Unchanged*	*Improved*	*Deteriorated*
0	8,103	NA	NA	NA	24,388	NA	NA	NA
≥ 1	3,645	1,886 (51.7%)	1218 (33.4%)	541 (14.8%)	5,866	5,511 (93.9%)	221 (3.8%)	134 (2.3%)
1	2,369	1,340 (56.6%)	**714 (30.1%)**	315 (13.3%)	4,391	4,179 (95.2%)	**136 (3.1%)**	76 (1.7%)
2	774	349 (45.1%)	**294 (38.0%)**	131 (16.9%)	1,058	969 (91.6%)	**54 (5.1%)**	35 (3.3%)
≥ 3	502	197 (39.2%)	**210 (41.8%)**	95 (18.9%)	417	363 (87.1%)	**31 (7.4%)**	23 (5.5%)

Table 8: The results of Cppcheck v1.86 and Cppcheck v2.13 on different versions of the SOTorrent dataset.

	Answer #	*Code Snippet #*	*Code Version #*
	SOTorrent18		
Cppcheck *v*1.86	15,724	16,533	20,664
Cppcheck *v*2.13	23,253	24,699	30,923
	SOTorrent22		
Cppcheck *v*1.86	19,485	20,450	25,832
Cppcheck *v*2.13	28,521	30,254	38,248

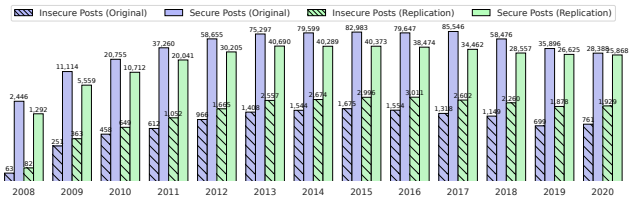


Figure 9: Yearly distribution of secure and insecure posts discovered by DICOS (logarithmic scale) as reported by the authors (Figure 6 in [37]) and found in our replication study.

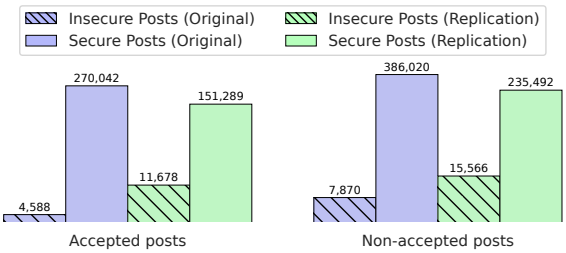


Figure 10: Ratio of insecure accepted and non-accepted posts discovered by DICOS (logarithmic scale) as reported by the authors (Figure 7 in [37]) and found in our replication study.

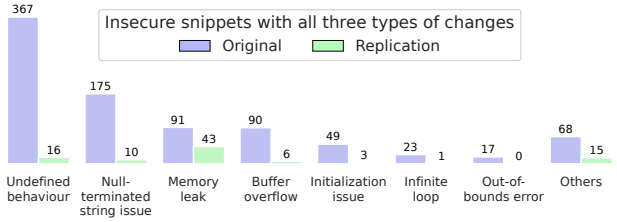


Figure 11: Types of discovered insecure code snippets with three types of changes discovered by DICOS as reported by the authors (Figure 8 in [37]) and our replication study.

from 0.0038 to 0.05, all p-values < 0.05). This suggests a consistent and significant disparity in the proportion of insecure posts across the entire study period.

Figure 10 compares the original results and the replication findings. As explained in §6.2, we found a higher ratio of insecure posts in both accepted (1.67% ↗ 7.72%) and non-accepted answers (1.99% ↗ 6.61%). Nevertheless, the original conclusion that secure posts outnumber insecure posts in both categories still remains valid. A two-sample z-test for proportions of the insecure to secure posts ratio between the original and our replication results shows a significant difference ($Z = -126.888$, $p < 0.001$).

Figure 11 presents a comparison between the authors' original results and the replication findings for the types of insecure code snippets with all three types of changes (i.e., changes in security-sensitive APIs, security-related keywords, and control flows). Overall, there is a significant decrease in the number of snippets with all three change types, for instance, *Undefined behavior* (367 ↘ 16), *null-terminated string issue* (175 ↘ 10) and out-of-bounds error (17 ↘ 0).