Measuring the Effects of Stack Overflow Code Snippet Evolution on Open-Source Software Security

Alfusainey Jallow, Michael Schilling, Michael Backes, Sven Bugiel
CISPA Helmholtz Center for Information Security, Germany
{*alfusainey.jallow, schilling, backes, bugiel*}*@cispa.de*

***Abstract*—This paper assesses the effects of Stack Overflow code snippet evolution on the security of open-source projects. Users on Stack Overflow actively revise posted code snippets, sometimes addressing bugs and vulnerabilities. Accordingly, developers that reuse code from Stack Overflow should treat it like any other evolving code dependency and be vigilant about updates. It is unclear whether developers are doing so, to what extent outdated code snippets from Stack Overflow are present in GitHub projects, and whether developers miss security-relevant updates to reused snippets.**

To shed light on those questions, we devised a method to 1) detect outdated code snippets versions from 1.5M Stack Overflow snippets in 11,479 popular GitHub projects and 2) detect security-relevant updates to those Stack Overflow code snippets not reflected in those GitHub projects. Our results show that developers did not update dependent code snippets when those evolved on Stack Overflow. We found that 2,405 code snippet versions reused in 2,109 GitHub projects were outdated, with 43 projects missing fixes to bugs and vulnerabilities on Stack Overflow. Those 43 projects containing outdated, insecure snippets were forked on average 1,085 times (max. 16,121), indicating that our results are likely a lower bound for affected code bases. An important insight from our work is that treating Stack Overflow code as purely *static* code impedes holistic solutions to the problem of copying insecure code from Stack Overflow. Instead, our results suggest that developers need tools that continuously monitor Stack Overflow for security warnings and code fixes for reused code snippets and not only warn during copy-pasting.

1. Introduction

The work of software developers is strenuous. To ease their job, developers seek assistance in different forms, such as better IDE support, reusable code, accessible guidelines and best practices, or exchange with other developers. One internet information source has particularly become popular: Stack Overflow, the most prominent question-and-answer site for programmers to share and increase their knowledge. Stack Overflow is popular among developers because most answers provide a concise explanation and an example code snippet about how a particular technology should work. Given developers' time constraints and economic pressure, having ready-to-use and good functional code snippets is a welcome time-saver. As many recent studies show, this sentiment is common among developers and contributes heavily to Stack Overflow's unabated popularity [1]–[7].

Code on Stack Overflow is constantly evolving. The developer community continuously adds new snippets and maintains existing ones [1], sometimes addressing bugs and vulnerabilities. When app developers reuse code snippets from Stack Overflow, they effectively create a dependency on those code snippets. Thus, when a newer version of a reused snippet on Stack Overflow includes security and/or bug fixes, developers' negligence in updating their reused version renders their code unnecessarily vulnerable or buggy. Ultimately, this means that code on Stack Overflow needs to be treated like any other code dependency managed with some version control. However, past work by Manes and Baysal [8] has shown that code on GitHub, which attributes its origin to Stack Overflow, and the original code snippet on Stack Overflow evolve independently from each other. This suggests that developers do not track changes to reused Stack Overflow code. Hong et al. [9] investigated the change histories of Stack Overflow snippets to identify insecure code snippets. They found the first evidence that outdated, vulnerable snippets can be found in the current code bases of open-source C/C++ projects. Zhang et al. [10] detected vulnerable C/C++ snippets on Stack Overflow and found that snippet revisions are associated with reducing the number of code weaknesses.

These observations set the stage for our work. We are interested in a more comprehensive picture of the effects of Stack Overflow code snippet evolution on open-source software security, and we explore the following research questions: ***RQ1****: Do Stack Overflow code artifacts reused in developer code bases evolve on Stack Overflow?* ***RQ2****: Can we find evidence that developers monitor Stack Overflow for code updates?* ***RQ3****: Do developers miss security fixes on Stack Overflow for code also present in their code bases?*

To answer our research questions, we take inspiration from prior works, but we devise in this paper a methodology that addresses the shortcomings of those works. First, attributing Stack Overflow snippets when copying them to developer code bases is the exception and not the norm [11]. Thus, limiting the dataset to snippets of attributed posts provides only an incomplete, biased picture. Second, automatically classifying code snippets as vulnerable is an unsolved problem when not scoped to a narrow problem domain (e.g.,

crypto APIs [6]) and even state-of-the-art work [9] relies on heuristics and idiosyncrasies of a single programming language. Thus, the strategy to identify vulnerable snippets on a large scale and re-identify them in developer code bases is inherently limited by the classification tool (e.g., supported languages or accuracy). In our approach, we first used clone detection to identify all the versions of 1.5M code snippets from Stack Overflow (in total, 3.5M code snippet versions) in the code bases of 11,479 popular open-source projects on GitHub without limiting our dataset to only attributed snippets. We focused on code snippets in the most popular programming languages Python, Java, JavaScript, and C. From that, by comparing the change history of Stack Overflow snippets with the latest version of code on GitHub, we retrieve the set of code snippets that evolved on Stack Overflow since appearing in developer code bases, and that is hence outdated in the GitHub projects (RQ1). Additionally, applying clone detection to the entire change history of the GitHub code bases and that of Stack Overflow posts allows us to detect whether developers updated their code to a newer version of the Stack Overflow snippet, answering RQ2. Lastly, for RQ3, we used a combined method of natural language processing of commit messages and comments, analyzing code properties, and manual confirmation to determine which evolved code snippets on Stack Overflow contain security-relevant edits that have not been transferred to the code bases on GitHub.

We found that 2,405 code snippet versions reused in 22,735 distinct GitHub source files were outdated, affecting 2,109 GitHub projects. Among those outdated snippet versions, we manually verified 26 to have a security-relevant update on Stack Overflow that fixes a known vulnerability. The fixes to those vulnerabilities on Stack Overflow were not reflected in 43 highly popular, non-forked open-source projects to whose maintainers we disclosed our findings. Further analyzing the 43 affected projects reveals that it took, on average, 1,060±506 days (max. 3,303 days) from when an insecure Stack Overflow code snippet is committed to a GitHub project until the time a Stack Overflow comment raising a security warning is made. However, as soon as a security warning is raised, it takes, on average, 296±200 days (max. 1,516) for a security issue in a code snippet to be fixed. For a time difference this long, it would be non-trivial for developers to manually track Stack Overflow for updates or be aware of (or react to) security discussions around a piece of code they reused such a long time ago.

While prior work [12], [13] made crucial contributions in providing tools to help developers reuse Stack Overflow code snippets more securely, those works treated code on Stack Overflow as *static* and thus only helped to mitigate current instances of insecure code reuse. For instance, available tools can flag a code snippet as secure. However, as our results show, the developer community on Stack Overflow can raise a security warning about such a snippet months or even years later. And developers that reused the insecure version suddenly depend on an insecure code snippet. To our knowledge, no tool currently exists that tackles this problem of completely bridging the Stack Overflow and GitHub knowledge ecosystems. Our findings show that code on Stack Overflow is "evolving code." This observation calls for a different approach to building tools to tackle the problem of insecure code reuse from Stack Overflow. In addition to warnings during copy-pasting, tools should constantly monitor Stack Overflow for security warnings or code fixes and notify developers to reduce the amount of outdated insecure Stack Overflow snippets reused in production code.

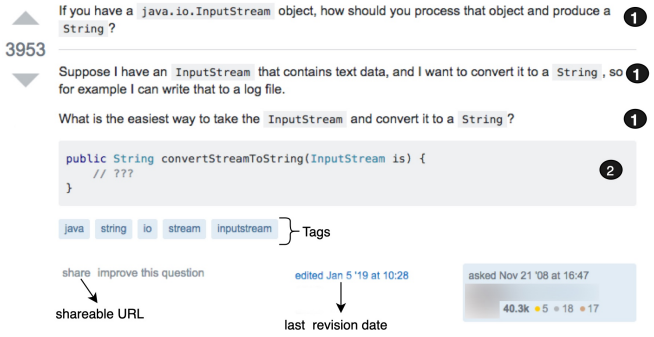


Figure 1: Question post that has 3,953 votes (score) and contains three text blocks (❶) and one code block (❷).

2. A Primer of Stack Overflow

A discussion thread (short: *thread*) on Stack Overflow usually consists of one or more *posts*. The first post in a thread is the question; all other posts are answers. Moreover, each post consists of text blocks and code snippets. A text block contains text written in a natural language and provides additional context for readers of a post. A code snippet contains a chunk of code written in a programming language. Figure 1 depicts a question post that contains three text blocks and a single code snippet.

A unique `PostId` identifies every Stack Overflow post. Using the PostId, it is possible to reference a specific post directly and to share direct access to it via URL. Some developers copy code snippets from a post and include the URL of the post in their source code files to make its Stack Overflow origin explicit. This way of referencing Stack Overflow posts in developer code bases—a behavior in line with the Stack Overflow license—is what we refer to as *attribution*. In addition, Stack Overflow provides a commenting feature for posts, which can give feedback on whether a code snippet in a post works, whether it contains a bug, a vulnerability, or another idea for a solution.

Version Control of Stack Overflow Artifacts: In general, posts on Stack Overflow are not static content but strongly influenced by the community's reaction (e.g., through comments or up/down votes), which in turn can motivate the creator of a post, or the developer community, to edit and expand the content of posts. For the creation of every post as well as for every post edit, Stack Overflow creates a post version object to record all text and code changes made, together with the date the change was made

Figure 2: A Stack Overflow answer post with comments and change history. The original post was created on Aug 12, 2009, and the code was edited on Feb 23, 2011. Almost ten years later, a comment (dashed line) points out an *XXE* vulnerability, which led to a code revision (see Figure 3).

and the author that made the changes. An optional commit message by the user summarizing the changes they made is stored alongside the post version. The post version object is then stored in a post change history, a collection of all the post versions. Stack Overflow provides version control for posts by recording all the edits made to posts using a post change history. Even though Stack Overflow has a comprehensive version control for posts, only the latest version of posts is displayed on the Stack Overflow website.

We briefly illustrate the versioning of Stack Overflow posts using a concrete example from our results. Figure 2 depicts an example answer post (id `1264912`) [14] that was created on Aug 12, 2009. In the last comment of this answer post (marked with a dashed line), posted almost ten years after version #3 of the post, a developer pointed out that the code snippet contains an *XML eXternal Entities* (XXE) injection vulnerability (CWE-611) and provided a link to the OWASP website detailing the nature of the vulnerability. This vulnerability, if not fixed, may allow an attacker to disclose confidential data or server-side request forgery. For this reason, the vulnerability is listed in the OWASP top 10 web application security risks. Figure 3 shows the last versions of the answer post. In version #4 of the answer, the vulnerability was fixed thanks to this comment, as also noted in the commit message of that version. In this code version, the changes include the fix recommended in the OWASP website by protecting the `TransformerFactory` Java object. In our results, we found the vulnerable version #3 of that code snippet in the current code base of the *Apache Chemistry* project and the popular *Apache Lucene-*

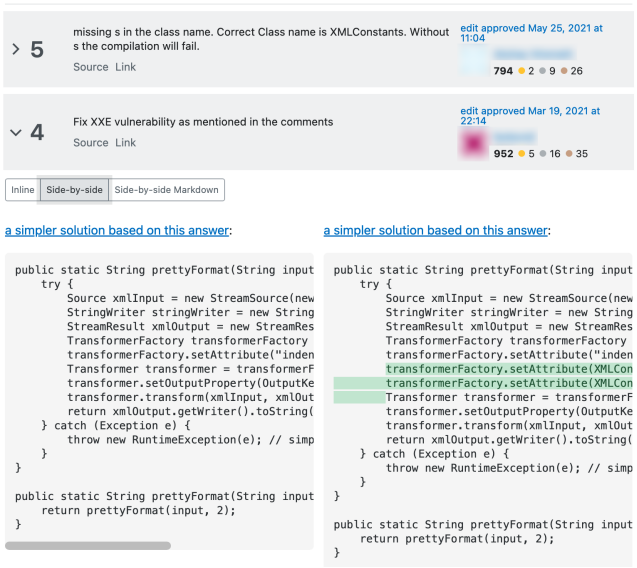


Figure 3: *Fourth version* of the answer post in Figure 2. Based on the pointed-out XXE vulnerability, the 4^{th} version of the post (on Mar 19, 2021) includes the corresponding fix, as also noted in the commit message of the version.

Solr project. The vulnerable snippet was committed to the Apache Lucene-Solr project on Jan 25, 2012, almost one year after version #3 was created and nine years before the vulnerability was fixed in version #4.

The evolution of artifacts on Stack Overflow is similar to how software evolves. A program evolves when bugs or vulnerabilities are fixed, new features are added, or code is updated due to changes in requirements. A version control system tracks changes to source code and other artifacts, while a bug tracking system is used to triage and solve bugs or issues. In this work, we transfer the concepts of bug tracking and version control from software projects to Stack Overflow. In the example above, the post change history provides version control for the code snippets, and the comments provide a form of loosely-coupled issue tracking.[1]

3. Related Works

We provide a brief overview of related works that studied code on Stack Overflow or assessed the problem of outdated third-party libraries, which carries a conceptual resemblance to the problem of using outdated code from Stack Overflow.

3.1. Stack Overflow

Fahl et al. [4], [5] studied developers who consulted Stack Overflow for help and found high-profile applications vulnerable to MitM attacks because developers copy code examples specifically disabling TLS functionality.

1. It is worth pointing out that the commenting feature is not a bug-tracking system per se. Only comments about a buggy or insecure code snippet are considered a bug report.

Acar et al. [3] studied which resources developers consult when facing a security-related problem. To this end, they conducted a lab study with 54 developers, providing them with different information sources to consult while implementing a security feature. They conclude that developers that use books and official documentation are more likely to write secure code. At the same time, those that consulted Stack Overflow are more prone to writing insecure but more functionally correct code.

Zhang et al. [15] studied security API misuse patterns and found that several Stack Overflow code examples containing improper API usage were reused in open-source projects. Verdi et al. [13] found 69 vulnerable Stack Overflow code snippets and usages of the vulnerable snippets in over $2,850$ projects on GitHub. The authors built a browser extension capable of detecting known vulnerable C++ code snippets to stop their further propagation.

Fischer et al. [6] investigated the number of insecure code samples on Stack Overflow that resurfaced in high-profile Android apps on Google Play. The authors used a learning-based approach to classify code snippets into secure or insecure, depending on the usage patterns of Java and third-party crypto APIs. They discovered that about 30% of the insecure code samples had been reused in security-critical apps. In a more recent study, Fischer et al. [12] added a warning to posts containing insecure code snippets to assist developers in avoiding them. Developers can either copy the insecure code or consult a list of recommended posts with similar functionality but secure code.

While those works made crucial contributions in pointing out the problem of vulnerable code snippets on Stack Overflow and their propagation to developers' code bases, none considered these code snippets as "evolving code" or studied the impact of that evolution on software security.

Zhang et al. [10] scanned 650k C/C++ snippets from Stack Overflow for 89 CWEs (Common Weakness Enumeration) and discovered 13k vulnerable snippets. The authors also investigated the code evolution history of C/C++ snippets and found that, in general, code revisions are associated with reducing the number of code weaknesses.

The recent *Dicos* paper by Hong et al. [9] is closest to our work. Dicos detects security fixes in the change histories of Stack Overflow posts by searching for security-relevant keywords in natural language texts and scanning for security-relevant changes to a snippet's control flow or security-sensitive APIs. Applying Dicos to 668k posts tagged with C, C++, or Android, the authors found 12k insecure posts. Using clone detection on 2k popular C/C++ open-source projects, the authors could further detect insecure snippets in 151 projects. While Dicos' results relate to our RQ3, i.e., if outdated insecure snippets are found in GitHub projects, Dicos' implementation is limited (e.g., it only compares the initial and the last version in the change history of code snippets) and not directly applicable for our methodology (e.g., insecure snippets must contain certain types of edits idiosyncratic to selected programming languages). We further discuss the applicability of Dicos for our methodology and differences in our work in §4.

Stack Overflow and GitHub have also been the subjects of measurement and developer behavior studies, e.g., [11], [16]–[23]. Closest to our approach, Manes and Baysal [8] compared the change histories of 23k GitHub projects and 4.6k Stack Overflow code snippets that were explicitly *attributed*. They found that reused snippets develop independently and that Stack Overflow snippets primarily evolve in their text blocks. The authors did not use code clone detection to determine if attribution was accompanied by copied code and relied entirely on time series analysis like impact latencies. They left the application of code clone detection open for future work. Further, Baltes et al. [1] created the *SOTorrent* dataset, which we also use in our work (see §4). Based on this dataset, the authors showed that code on Stack Overflow is maintained and does evolve.

3.2. Third-party Libraries

Outdated third-party libraries have been shown to have security implications for apps that rely on them. Derr et al. [24] found that vulnerabilities stayed longer in Android apps because app developers were slow in updating outdated library versions. Lauinger et al. [25] investigated the usage of JavaScript libraries in web apps and found that web developers do not react in time to update library versions to the most recent bug fix release.The authors also interviewed developers and discovered that many developers refrained from updates due to fear of incompatibilities, lack of necessity, or being unaware of updates.

By copying code from Stack Overflow, developers create a dependency similar to including a software library. Thus, there exist strong conceptual similarities. However, no work has studied whether the same problems with outdated libraries exist for outdated code snippets. In contrast to library detection, we face different challenges: First, we need to re-identify very small code pieces with just a handful of lines of code compared to complete libraries. Second, in contrast to libraries, there are no systems like Common Vulnerabilities and Exposures (CVE) to report vulnerable code snippets, but instead, we have to rely on the snippets, their commit messages, and surrounding discussions to classify snippets, similar to Dicos [9].

4. Effects of Code Snippet Evolution

We first present our methodology (§4.1) to answer our research questions: **RQ1:** *Do Stack Overflow code artifacts that are reused in developer code bases evolve on Stack Overflow?* **RQ2:** *Can we find evidence that developers monitor Stack Overflow for code updates and update their projects accordingly?* **RQ3:** *If developers miss updates to reused code snippets on Stack Overflow, do they also miss security-relevant updates?* Afterward, we present our results and findings (§4.2).

4.1. Research Methodology

Our data-driven approach to answering our questions is depicted in Figure 4. In general: We used *clone detection*

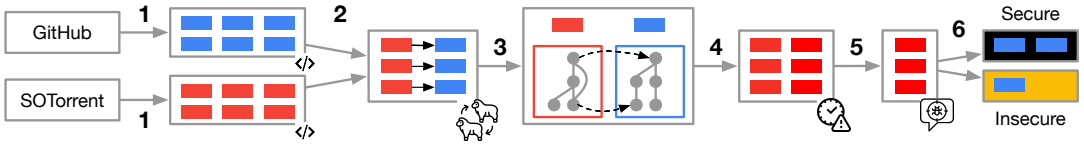


Figure 4: Our research methodology: (1) extraction of Stack Overflow code snippets and developer code from GitHub, (2) clone detection between Stack Overflow change histories and GitHub, (3) timeline analysis and filtering to identify best candidates for code clone pairs, (4) identifying outdated (RQ1) and updated (RQ2) snippets by extracting code evolution on Stack Overflow and comparing change histories between platforms, (5) filtering reused outdated Stack Overflow posts without comments and revisions without commit messages, and (6) classifying security-relevance of Stack Overflow code edits based on comments, commit messages, code changes, and manual confirmation for RQ3, detecting fixes to reused vulnerable code snippets which did not propagate to GitHub.

TABLE 1: Distributions of the code snippet data set from Stack Overflow showing each programming language's initial and final sample size. The final sample is selected by removing all single-version snippets and snippets containing at least one version with less than 10 LoC in their change history.

	Initial		**Final** (>1 Version $\cap >= 10$ LoC)						
					Versions				
Language	**No.**	ϕ **Score**	**No.**	ϕ **Score**	**No.**	**Min**	**Max**	ϕ **No.**	**Median**
Java	2,429,964	1.9	**524,099** (21.6%)	2.7	1,209,700	2	30	2.3	2
C	404,051	1.6	**112,138** (27.8%)	1.9	274,508	2	36	2.4	2
Python	1,580,623	2.0	**336,328** (21.3%)	2.9	807,345	2	26	2.4	2
JavaScript	2,704,287	1.8	**530,205** (19.6%)	2.9	1,227,326	2	743	2.3	2
Overall	7,118,925	1.9	**1,502,770** (21.1%)	2.8	3,518,879			2.3	

to find code in open-source projects that are highly similar to code snippets on Stack Overflow. For each occurrence of reused code, we compared the change histories of this code between the open-source projects and Stack Overflow. From this comparison, we identified instances of outdated Stack Overflow code snippets within open-source projects. In a final step, we analyzed the change history of the reused, outdated code snippets for security issues and corresponding fixes by mining bug-fix commit messages and using information from the discussions on Stack Overflow around corresponding posts, and then mapped those results to the open-source projects containing affected code snippets.

Overall, we carried out this analysis process for four different programming languages: *Java* and *C*, since they are the focus of the majority of prior security-related studies on Stack Overflow [3], [6], [9], [9], [11]–[13], [26], and *Python* and *JavaScript* as they are currently the most popular languages on Stack Overflow [27]. We now describe the used data sets and the individual steps from Figure 4.

4.1.1. Study data sets. We used two main data sources for our study—SOTorrent [1] and GitHub—which are also commonly used in related works.

Stack Overflow code snippets: Our data set of Stack Overflow code snippets is based on the *December 31, 2020* release of the SOTorrent data set [1]. This data set stores the entire change history of all text and code on Stack Overflow, which provides the necessary version control that we need to track and analyze changes to individual code snippets. We will refer to a *snippet change history* when we mean the entirety of all versions of a code snippet.

Since SOTorrent does not contain information about the used programming language, we used Guesslang [28] to determine the programming language for the most recent versions of all code snippets with at least 5 LoC. Out of $31,287,646$ code snippets in the data set, Guesslang identified the language of $31,202,349$ code snippets. No assignment was possible for the remaining $85,297$ code snippets, mainly because the code snippets do not contain actual code. Our data set contains code snippets in 30 different programming languages.

A close examination of this data set showed that in line with prior work [16] many short code snippets consisted of highly trivial code, such as boilerplate code. Given that the first step in our study is to match Stack Overflow code snippets with code from open-source projects using clone detection, this would lead to a very high number of positive matches from whose analysis no meaningful information could be drawn. To reflect this fact, we limited our data set to only those snippets where each version in the change history consists of at least 10 LoC. Further, since our study focuses on code evolution, we excluded all code snippets that have never undergone any changes since their creation. We considered all code snippets that meet these criteria for clone detection and disregarded any other factors, such as post popularity, post views, or the reputation of users.

Table 1 provides an overview of the corresponding sample sizes for the four considered programming languages. We considered 1.5M code snippets, where most are written in Java (524k; 34.9% of final sample size) or JavaScript (530k; 35.3%). Those snippets have 3.5M versions, whereas the average snippet in our final data set has 2.3 versions. The average post in our initial data set had a score of 1.9, and the average post in our final sample had a score of 2.8,

indicating that posts with revisions have higher scores.

Open source projects: We used GitHub for our data set of open-source projects, as it is one of the most popular code hosting platforms in open-source communities [29]. Code on GitHub is version controlled using the Git version control system, allowing us to measure historical code overlap between Stack Overflow and GitHub. For our sample, we focused on popular and well-maintained projects, as we consider the overall impact of missed security-critical updates to be the highest and because we assume that the code base is steadily maintained over a longer period.

Using the GitHub Search API, we retrieved a list of projects sorted by popularity (number of stars) for each of the four programming languages considered in this study. We then manually inspected each project in this initial candidate list and excluded: a) *forks* to avoid including essentially identical code in the analysis; b) *archived* (retired) projects as it is not reasonable to expect the maintainers to provide a bug fix in case they are notified; and c) projects that are known to be learning materials[2] because the focus of our study lay on real software projects. Our final GitHub sample consisted of 11,479 individual projects (Java: 2,290, C: 2,241, Python: 3,107, JavaScript: 3,841) containing 4,098,397 source files.

4.1.2. Clone detection. To identify shared code between our GitHub and Stack Overflow samples, we used clone detection to search for complete or partial matches between all of the source files in the current versions of the open-source projects and the complete change histories of code snippets from Stack Overflow. As a first step, we only considered the most recent version of the open-source projects since we were interested in projects that currently have a dependency on Stack Overflow code snippets. The result is a list of all the source files code with Stack Overflow in common. In a second step, we then rerun the clone detection for the files in this list, but this time for all previous versions stored in the Git version history of a project—this is a *cartesian-product clone detection* between the change histories of GitHub projects and their Stack Overflow dependencies.[3]

To account for the differences in programming languages, we used two different clone detection tools to identify common code: *NiCad* [30] for Java and C, and *PMD Copy-Paste Detector* [31] for Python and JavaScript. Both tools have been fine-tuned with parameters specific to Stack Overflow code snippets to achieve the best possible results. For PMD CPD, we adopted the fine-tuned parameters from Baltes et al. [11], which were already optimized for usage with Stack Overflow code. For NiCad, we fine-tuned the parameters ourselves (see Appendix A). It is important to note that PMD CPD can only recognize exact copies of Python and JavaScript code and only returns exactly matching sections in source files. NiCad also recognizes slightly modified copies and returns a similarity score between GitHub file sections and code snippets when this score exceeds 83% (the cutoff is based on our calibration process).

We identified 108,134 file sections in 3,439 GitHub projects where code common with *multiple* snippet versions from Stack Overflow were found. A section of a GitHub file is defined by the triple 1) start and 2) end line in 3) a specific commit. For most file sections, we found several Stack Overflow code snippet versions with (nearly) identical code, which aligns with findings of prior work [22] on code duplication on Stack Overflow. On average, for every single Python source file section, we found the same code in 4.81 code snippets (max. 68 snippets), for Java in 3.19 snippets (max. 67), for C in 1.33 snippets (max. 26), and for JavaScript in 2.97 snippets (max. 46). We found 8,538 distinct reused code snippets with together 16,479 versions (from 8,436 distinct posts), where the average file section had 3.22 snippets (and 6.55 versions) as matching clones.

4.1.3. Identifying best candidates. Not all of the ≈16k snippet versions from the 8.5k snippets are relevant to our research questions. Since we are interested in measuring the impact of reused *outdated* snippets from Stack Overflow on the security of open-source GitHub projects, we can filter snippets that do not fit this setting. To this end, we implemented a *filter pipeline* that gradually reduced the number of candidate snippet versions per GitHub file section in our clone detection results. Table 2 details the filtering process for the clones to our initial 108,134 distinct file sections. The input to each filter step is reported as a count of *distinct* posts, snippets, and snippet versions. The average posts/snippets/versions per file section provide an overview of the *1:n* mapping between a single file section and detected clones on Stack Overflow. In cases where this relation becomes *1:0*, the file section is removed. In the following, we describe the individual filter steps.

First, we apply filters that are motivated by prior studies on the behavior of developers using Stack Overflow:

F1. Attributions Filter: Baltes et al. [11] have shown that some developers copy code from Stack Overflow and attribute the Stack Overflow source, a behavior in line with the Stack Overflow license. Accordingly, this filter relies on attribution as the strongest indication of code copy from specific Stack Overflow posts. If a section of a file or its immediate surrounding context includes attribution to a post on Stack Overflow, this filter excludes all snippets that do not originate from the corresponding post. In cases where a specific answer or question is not referenced (i.e., a Stack Overflow thread is attributed), all but the snippets belonging to the question and all answers in the thread are excluded.

F2. Commit Date Filter: Prior works [4], [6] have shown that developers seemingly copy code from Stack Overflow to their projects, and we apply this same assumption for the reduction of our data set. A logical prerequisite for a causal link between a Stack Overflow post and a GitHub source file is that the post existed before the code was committed on GitHub. Accordingly, this filter compares the change

2. Examples of such projects: https://github.com/topics/awesome

3. A cartesian-product clone detection between all versions of all Stack Overflow code snippets and the entire change history of all source files in our GitHub projects sample was not feasible. Already our step-wise approach took around 17 weeks using four servers (each with 64-core Intel Xeon E7-8867 and 1.5 TB Ram).

TABLE 2: The number of *distinct* posts, code snippets, and snippet versions for each filter step, the average no. of posts, snippets, and snippets version for each distinct detected GitHub file section, as well as the *total* count of posts, snippets, and versions removed by each filter.

	Input to filter (Distinct)				Average per GitHub file section			Removed by filter (Total)		
Filter	**File sections**	**Posts**	**Snippets**	**Versions**	ϕ **Posts**	ϕ **Snippets**	ϕ **Versions**	**Posts**	**Snippets**	**Versions**
F1: Attribution	108,134	8,436	8,538	16,479	3.2122	3.2152	6.5477	1,020	1,026	2,069
F2: Commit Date	102,236	8,118	8,217	15,827	3.1625	3.1656	6.4022	3,535	3,576	7,076
F3: Version Similarity	84,542	6,407	6,475	12,294	2.8375	2.8406	5.2595	0	0	5,673
F4: Snippet Similarity	84,542	6,407	6,475	7,086	2.8375	2.8406	2.8406	665	671	683
F5: Post Type	84,542	6,054	6,117	6,712	2.8176	2.8206	2.8206	1,151	1,155	1,190
F6: Post Popularity	84,542	5,237	5,296	5,850	1.6471	1.6500	1.6500	1,295	1,301	1,349
F7: Latest Post	84,542	4,336	4,389	4,880	1.0379	1.0405	1.0405	162	162	163
Final	84,542	4,241	4,272	4,755	1.0000	1.0000	1.0000	–	–	–

histories of the code snippets on Stack Overflow and the file sections on GitHub. It excludes all snippets created after the commit that introduces a snippet into a source file.

The number of clone Stack Overflow candidates per GitHub file section after **F2** still poses a severe challenge for the next steps in our analyses to determine the security impact of reused code. Ideally, we could classify all remaining ≈12k reused Stack Overflow snippets in our data set as security-relevant or not. However, during the project, it became evident that this type of analysis can only be partially automated with current tools and still requires much manual verification work. Although there are already promising approaches [6], [9], [12] for automating this task, these approaches have proven to be too limited in scope (e.g., works only with one programming language or a narrow aspect of security) or unsuitable for our current situation (see §4.1.5 for details). For this reason, we added additional filtering steps to reduce the number of matches between GitHub file sections and Stack Overflow snippet versions to a feasible amount for manual analysis.

The additional filters follow the basic idea that if a developer has reused code from Stack Overflow, it is rather unlikely that they have taken code from several different Stack Overflow snippets or different versions of them. Hence, this approach requires making several assumptions about how developers use Stack Overflow and comes with the limitation that the mapping may be subject to error. Despite these limitations, we believe this approach is currently the only practical solution allowing us to perform subsequent analyses of security relevance. When creating the additional steps of the filter pipeline, we oriented ourselves to first go through steps that leave the least room for interpretation.

In the next steps, we prioritized clones with higher similarity over those with lower similarity. The intuition is that the less similar the Stack Overflow snippets and GitHub code are, the less likely they are directly related. As PMD CPD only returns exactly matching sections and snippets, these filters are ineffective for Python and JavaScript.

F3. Version Similarity Filter: If multiple versions of the same snippet are found in a file section, this filter excludes all versions except for the one(s) with the highest similarity. If multiple versions have an equally high similarity score, the filter selects the latest version, ensuring the most conservative result in our later analysis.

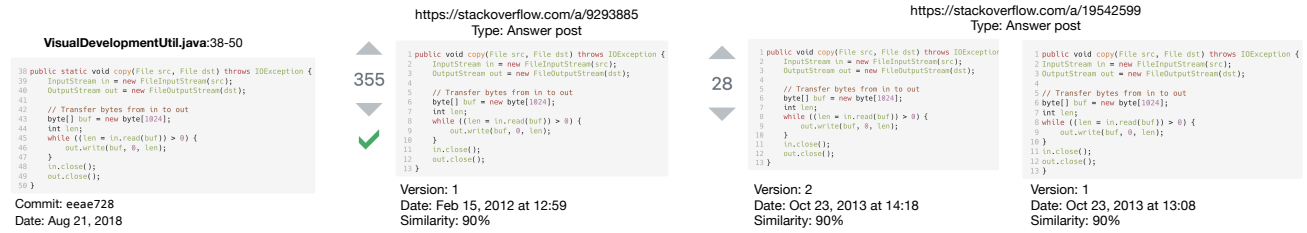
F4. Snippet Similarity Filter: This filter compares the similarity scores of the snippets for each file section, excluding all except the one(s) with the highest scores.

Lastly, if the above filters did not yield the best candidate version, we added filters considering developers' most likely behavior while using Stack Overflow.

F5. Post Type Filter: Following Baltes et al. [11], developers are more likely to reuse code from answer posts than question posts. Accordingly, if a file section contains clones of snippets from answers, this filter excludes all snippet clones from question posts from further consideration.

F6. Post Popularity Filter: This filter results directly from the nature of the Stack Overflow website. By default, Stack Overflow users are shown answers within posts sorted according to their score, which is a measurement of usefulness by other members of the Stack Overflow community. In line with other review systems (e.g., app stores and online markets), we assume developers will use this community-driven feedback when selecting information for their current programming task. Accordingly, we filter code snippets based on the popularity of their respective posts and select the code snippets belonging to the most popular post. However, a post might be popular for various reasons that cannot be easily determined automatically. To better consider the relevance of a post for a detected clone, this filter first computes the lines of code (LoC) of all the code snippets reported as clones and the file section containing the reported clones. If code snippets have the same LoC as the GitHub file section, the filter selects only the posts of those code snippets for comparing popularity. This means we assume at this point that developers rather copy entire code snippets than cherry-pick parts of them. If no such snippet exists, the filter selects the most popular one(s) from all posts.

F7. Latest Post Filter: This is the final filter in the pipeline, and at this point, all the remaining clones belong to the same post type (question or answer) and are equally popular. If the remaining clones belong to question posts, this filter selects the snippet belonging to the newest post. Otherwise, if the



(a) File section on GitHub (b) Stack Overflow post 9293885 (c) Stack Overflow post 19542599

Figure 5: Example for the execution of the filter pipeline. Filter **Version Similarity (F3)** removes version 1 of post 19542599. Filter **Post Popularity (F6)** selects revision 1 of post 9293885 as the final code snippet.

clones belong to answer posts and there is a single accepted answer, the code snippet in the accepted answer is selected. If there is more than one accepted answer, the code snippet belonging to the newest accepted answer is selected.

Example: To illustrate our filter pipeline, we go through the procedure for a concrete example using Figure 5. Figure 5a shows a code section from the Apache NetBeans project on GitHub—lines 38–50 of `VisualDevelopmentUtil.java` source file. Figures 5b and 5c show one, respectively, two matching Stack Overflow code snippet versions from two different answer posts. For the snippet in Figure 5c, the clone detector found two different versions (version 1 and 2) that match the file section in Figure 5a. Our filter pipeline sequentially determines the best candidate version among those three candidates as follows: The *attributions filter (F1)* will not discard any clones because the GitHub source file contains no attributions to Stack Overflow. Commit `eeae728` introduced the reused code on Aug 21, 2018, after any of the three snippets were posted (2012 and 2013, respectively). Thus, the *commit date filter (F2)* does not remove any candidates either. Next, the two versions of the snippet in Figure 5c have an equal similarity of 90% with the file section. Hence, the *version similarity filter (F3)* will select version 2, since it is the newest one, and discard version 1, reducing the overall set of candidates to two versions from two code snippets. Since those two versions have the same similarity of 90%, the *snippet similarity filter (F4)* cannot reduce the candidate set further. Similarly, both code snippets are in answer posts—post *9293885* [32] and *19542599* [33]—consequently, the *post type filter (F5)* will not remove any snippet. Next, the *post popularity filter (F6)* filter first determines that both snippets have the same LoC (12) as the file section and, thus, selects the more popular snippet, which is the snippet in Figure 5b with a score of 355 in contrast to the other snippet with a score of 28. Since this filter already reduced the candidate set to one, the *latest post filter (F7)* had no effect. As a result, we consider only the post and code snippet in Figure 5b in the further analysis steps.

Final data set: As depicted in Table 2, after the filtering pipeline, we ended up with 4,755 distinct snippet versions from 4,272 snippets (*JavaScript*: 1,979; *Java*: 909; *Python*: 1,112; *C*: 272) from 4,241 posts that were reused in 84,542 distinct file sections in 2,824 GitHub projects. Thus, we identified reused Stack Overflow code snippets in almost 25% of all 11,479 investigated GitHub projects. Based on this final data set, we answer our research questions.

Filter pipeline evaluation: To understand whether the filter pipeline approximates developer behavior, we evaluated it using a ground truth dataset comprising code snippets copied from *SO* to *GH*. Since it is not trivial to determine the exact source a piece of code is copied from (see §5.2), we relied on *attribution* since it is the strongest indicator of code reuse from *SO* [8], [11], [13]. We collected all source files from our data set of detected clones containing an attribution to answer posts since developers copy code from attributed answers [11]. Overall, 292 source files attribute 37 distinct answer posts. Those source files contain 315 file sections for which we detected code clones in 219 distinct posts (*JavaScript*: 97; *Java*: 79; *Python*: 41; *C*: 2). Thus, 182 posts were also reported as clones but were not attributed. This enables us to determine whether, for each of the 315 file sections, the filter pipeline *without* the *Attributions Filter (F1)* filter will select clones from the 37 attributed posts or the 182 unattributed posts.

The filter pipeline with only filters **F2** through **F7** selected the code snippets from the attributed posts in 304 (96.5%) cases. Only in 11 cases did the filters not select code snippet clones from the attributed posts. In 9 of these 11 cases, source files contained attributions to several *SO* posts. This may indicate that the filter pipeline makes the wrong judgment in cases where a source file attributes multiple *SO* posts in direct proximity to reused code snippets. These results show that the filter pipeline reflects the behavior of developers reusing code from *SO*.

4.1.4. Determining Outdatedness. Using our filtered data set, we answer **RQ1** and **RQ2**. To determine whether a clone of a snippet version is outdated or not, we check whether the *reused version* on GitHub is the latest version of the snippet on Stack Overflow. If the reused version is not the latest, the snippet evolved on Stack Overflow, rendering the reused GitHub version outdated. Otherwise, the copied version is up-to-date. Answering **RQ2** requires tracing the evolution of copied Stack Overflow snippet versions in GitHub source files. To determine whether an update to a snippet on Stack Overflow also transferred to the reused code snippet on GitHub, we look backward at the change histories of GitHub

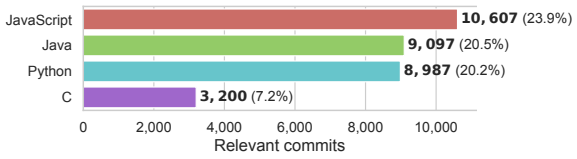


Figure 6: Distribution of *potential* security-relevant commit messages for different languages based on keyword-search.

files and Stack Overflow code snippets. We leverage the result of our cartesian-product clone detection to detect whether an older version of the snippet in a GitHub file matches an older version of the Stack Overflow snippet, indicating that the GitHub project developer became aware of the change on Stack Overflow (e.g., by monitoring the post, potentially proposing the edit on Stack Overflow or other channels like independent code origins).

4.1.5. Finding Security Fixes. To answer **RQ3** as to whether developers miss security-relevant updates on Stack Overflow for reused outdated code, we needed to find a way to filter out the large fraction of inconsequential code changes on Stack Overflow [34] and to determine whether an update to a code snippet version fixes a security issue. However, determining whether a code edit fixes a security issue is a non-trivial task, especially for incomplete/not self-contained code snippets as found on Stack Overflow. To the best of our knowledge, there are no generic, automated tools capable of determining whether or not a code update fixes a security issue. Prior studies on Stack Overflow with the need to classify the security of code snippets are either too narrowly focused [3], [6], [26]—for example, only misuse of Java crypto APIs—or work only with specific programming languages [6], [12], [13], [26].

The closest to a generic solution for classifying the change histories of Stack Overflow snippets is Dicos [9] (see also §3). Using keyword search in comments as well as changes in code snippets, Dicos detects security-relevant changes to snippets. Unfortunately, Dicos' implementation has limitations that prohibit an application in our study. First, it only considers the changes between the first and the last version but not between intermediate versions, i.e., pinpointing the exact Stack Overflow code snippet versions that are vulnerable and bug-fixed. This prohibits the detection of outdated-but-secure snippets in GitHub files (RQ3) and prevents the detection of bug fixing in GitHub projects (RQ2). Second, Dicos' classification requires certain code edits—changes to control flow and security-relevant APIs—which are tailored to the idiosyncracies of C/C++ and further narrow the scope of detection.

Thus, without an automated, suitable classifier for code snippets, we used keyword search in comments (inspired by Dicos) and mining commit messages to find bug-fix commits. The software engineering community has extensively studied the mining of commit messages to find bug fixes. We leveraged the approaches by Pan et al. [35] and by Osman et al. [36] to find potential bug fixes in Stack Overflow code edits. Figure 6 provides a quick overview of *potentially* security-relevant commit messages in *all* commit messages in SOTorrent based on the keywords from those prior works.

Since comments and commit messages are natural language texts, a possible approach could be to build an NLP classifier capable of determining whether or not a given sentence raises a bug report or indicates a security fix. However, the only data set of labeled code snippets is by Fischer et al. [6] and is limited to crypto API misuse in Java code. A first experiment based on this data set showed that the resulting detection of *generic* security-relevant text with a classifier trained on this data set performed poorly. Thus, we decided to apply approaches that use keyword search and manually verify security-relevant comments, commit messages, and code edits—which forms a data set for future research on the security of Stack Overflow snippets.

4.1.6. Manual Verification of Security-relevant Edits. The final step of our methodology is verifying whether comments, commit messages, and code snippet edits are security-relevant. Two researchers went through the list of all potential security-relevant changes and manually classified them according to the following three categories:
Security-relevant: A fix in a snippet is security-relevant if the issue fixed has a known weakness to which a Common Weakness Enumeration (CWE) [37] identifier is assigned. The assignment to CWEs is based on a combination of the surrounding context (comments and commit messages) and the *diff* in the post version that fixed the issue.
Bug Fixes: Fixes in this category are general software bugs with no CWE identifiers assigned and, hence, no direct security implication. *Undefined Behaviour* is one of the software bugs in this category. These are specific errors that produce unexpected results in an application (e.g., missing null checks or incorrect arithmetic calculations).
Improvements: Fixes in this category concern edits to reused code snippets that are not in the above categories. Examples include performance improvements, adding new language features, or removing deprecated APIs.

Based on this classification, we answer our **RQ3** whether an open-source contributor missed a security-relevant change on Stack Overflow since the time a reused code snippet was introduced in the open-source project.

4.2. Results and Findings

4.2.1. Projects missing updates on Stack Overflow (RQ1). Table 3 summarizes the file sections containing outdated versions of Stack Overflow code snippets. Overall, we found that 2,405 snippet *versions* (51% out of 4,755 reused ones) from 2,305 snippets were outdated, and we found them in 38,623 (46% out of 84,542) file sections on GitHub. Those outdated snippets come from 2,290 distinct Stack Overflow posts and are reused in 22,735 distinct source files from 2,109 distinct GitHub projects. In other words, almost every fifth (2,109 out of 11,479) project in our data set contains *outdated* code from Stack Overflow. Considering

TABLE 3: Summary of *distinct* projects and file sections containing outdated code snippets reused from Stack Overflow, grouped by programming language.

	Outdated Versions	Affected File Sections	Affected Projects
C	155 (53%)	894	**195** (70%)
Java	417 (44%)	1,380	**426** (62%)
Python	628 (51%)	6,938	**549** (74%)
JavaScript	1,205 (53%)	29,411	**939** (84%)
Total	**2,405** (51%)	**38,623** (46%)	**2,109** (75%)

only the 2,824 projects with reused code, almost 75% of them contained at least one outdated snippet.

The average affected project had 11.01±5.83 source files with outdated snippets, where JavaScript projects have the highest average number of affected source files (19.13±12.96), and Java projects the lowest number on average (2.69±0.65). Comparing the number of distinct outdated snippet versions (2,405) and affected files (22,735) shows that snippets commonly appear in multiple places in affected projects. The average outdated code snippet *version* in our data set has been reused in 12.14±7.64 *distinct* source files. Snippet `184588891` is the most reused snippet in our data set and was reused in 7,088 distinct source files (and 9,592 times in total). Comparing the number of distinct posts where those outdated snippets have been posted with the number of distinct snippets shows that 15 snippets appeared in the same post as another snippet and that 14 posts contained multiple outdated reused snippets. We provide further results in Appendix C.

4.2.2. Detecting code snippet updates (RQ2). After examining the cartesian-product clone detection between change histories of GitHub files and Stack Overflow snippets that were reused in those files, we did not find any evidence for updates in any of the projects, i.e., the reused snippet versions were constant in the source file since the origin commit in which the snippet was introduced.

> We found every second reused snippet to be outdated, with no differences between snippets from different languages. This affected every fifth popular open-source project at least once. Further, we did not find any indication that GitHub developers transferred updates to reused Stack Overflow code from Stack Overflow to their code bases. Thus, code snippets seem to be at most up-to-date until a code snippet evolved and never again afterward. This indicates that developers do not track snippets copied from Stack Overflow for changes or are unaware that the code they reused is being discussed and updated/fixed on Stack Overflow.

4.2.3. Security updates on Stack Overflow (RQ3). Using our keyword search approach, we found that 505 posts (22% out of the 2,290 posts linked to the outdated GitHub projects) contain potential security-relevant discussions or code fixes. Together, these posts comprise 278 answers and 227 questions. By *manually validating* each of the 227 question posts, we discovered that only a single question post contains a fix to an insecure code. This fix did not propagate to the AppScale GTS project, to which the insecure version of the snippet was copied. For the 278 answers, we found 25 containing genuine security/bug fixes. The remaining question and answer posts were either not security-relevant or developers copied the patched snippet version.

We found 26 posts containing fixes to 15 distinct security issues missing in 43 (2% out of 2,109) distinct GitHub projects. Table 4 lists the types of issues that we found, grouped according to their vulnerability category and referenced by their CWE identifiers. Since a project might be affected by multiple vulnerability types and posts might contain multiple vulnerabilities, we also report the count of distinct posts and projects. General bug fixes (no CWE assigned) affected 15 distinct projects, while 38 distinct projects copied snippets with weaknesses that have security ramifications. We found multiple vulnerability types in three projects, ngrok-libev (3), hashkill (3), and Mirai-Source-Code (2), and we found one answer post (id `779960`) that contained three vulnerabilities (CWE-754/-835/-20).

The most severe vulnerabilities—-according to OWASP—-that we found are XML eXternal Entities injection (CWE-611), Buffer Overflow (CWE-120), and Improper Input Validation (CWE-20) vulnerabilities. The most reused, outdated, insecure code snippet is from answer post `122721` [38], which suffers from an Incorrect Type Conversion (704) issue (see Figure 7). The code snippet containing the issue was posted in version #1 of the answer on Sep 23, 2008, and was reused in 4 open-source projects. Eight years later, a developer pointed out in the comments on Sep 18, 2016, that the code snippet will invoke undefined behavior if the necessary cast is not added to the `isspace` library function. The issue was subsequently fixed 20 days later. However, this fix did not propagate to the four open-source projects that reused the insecure version. We provide further examples in Appendix F.

Table 10 in Appendix E shows the 43 affected GitHub projects, which were "watched" on average 157.4 times (max. 1,494), received an average of 4,089.63 stars (max. 70,614), and were forked on average 1,085.4 times (max. 16,121). Considering this high number of forks, the vulnerabilities and bugs we found might have propagated on GitHub, and the 43 projects affected are likely a lower bound. Besides missing security fixes, we found another 20 posts containing fixes to improvements that were not reflected in 26 GitHub projects (see Appendix D).

> Though the number of posts we verified as true positives to contain a security weakness is low, we re-identified those snippets in a noticeable number of highly popular open-source projects with thousands of forks and stars. Given that we only studied non-forked projects, this popularity can amplify the impact of weaknesses in origin repositories.

Incident timelines: Based on the projects with missing

TABLE 4: Distinct security/bug issues found in Stack Overflow snippets, whose fixes on Stack Overflow are not reflected in 43 distinct GitHub projects that contained vulnerable versions of code snippets from Stack Overflow

CWE		Lang.	#Posts	#Count	#Projects
Security-relevant Weaknesses					
120	Buffer Overflow	C	1	1	1
172	Encoding Error	Python	1	1	1
754	Improper Check for Unusual or Exceptional Conditions	Python	1	1	1
		C	1	2	2
		JS	1	2	1
20	Improper Input validation	C	2	2	2
404	Improper release of resource	Java	3	6	6
704	Incorrect Type Conversion or Cast	C	2	5	5
835	Infinite Loop	C	1	2	2
1339	Insufficient Precision	JS	2	4	3
772	Missing release of resource	Java	1	1	1
		Python	1	1	1
690	Unchecked Return Value to NULL Pointer Dereference	JS	1	2	2
475	Undefined Behavior for Input to API	Java	1	2	2
194	Unexpected Sign Extension	C	1	1	1
611	XML eXternal Entity (XXE) Injection	Java	1	2	2
General Bugfixes					
	Undefined behaviour	Python	2	2	2
		Java	1	2	2
		JS	1	1	1
	Others	Java	2	2	2
		Python	1	9	8
Total (Distinct)			**28 (26)**	**51**	**48 (43)**

security fixes, we attempt to understand potential developer behavior better. We base this analysis on the order of events when reusing code from Stack Overflow in GitHub projects and the intervals between those events. Figure 8 depicts the possible timeline of events when a developer commits a clone of an insecure code snippet to a GitHub project after it was posted on Stack Overflow (t_{posted}). The commit can happen either before a security-relevant comment is made in the Stack Overflow post (t_{clone_before}) or afterward (t_{clone_after}). As a consequence of this comment ($t_{comment}$), the code snippet is revised to fix the issue ($t_{revision}$). We constructed this timeline for all open-source projects missing security fixes. If multiple comments raise the same issue, we select the earliest comment to understand whether (or not) an indication of a security issue with the code snippet was present when the snippet was reused.

For 12 of the 26 posts with fixes, the security issue was only raised in the commit message of the revision. For the remaining 14 posts, the issues were raised in the comments, and we consider only those 14 posts at this point. Of those 14 posts, 15 distinct snippet versions were reused in 25 source files of 24 GitHub projects. We analyzed the timeline of 29 incidents where vulnerable code with a CWE was reused and a security-relevant comment was posted.

Table 5 summarizes our analysis results. The reused code was committed in 17 of the 25 cases before the first security-

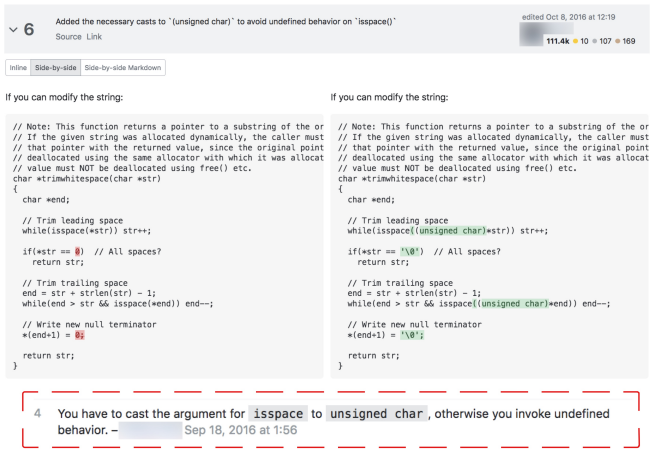


Figure 7: Answer 122721 containing a call to the *isspace* library function without the necessary cast to *unsigned char* (left-hand-side), which was pointed out in the comments (bottom) and subsequently fixed 20 days later (right-hand-side), as also noted in the commit message of the revision.

Figure 8: Timeline of events between two versions of a post.

relevant comment appeared. On average, developers reused a code snippet ≈1.5 years (556 days) after it was posted on Stack Overflow, and a security-relevant comment appeared on average ≈2.9 years (1,060 days) after the commit. There could be various reasons for this long interval before a comment, e.g., initial unawareness about vulnerabilities or delayed comments by security-savvy users. Consequently, developers must track Stack Overflow threads with reused code for a long time to avoid missing relevant comments.

For the remaining 12 cases, a security-relevant comment was available when the code snippet was committed to GitHub, i.e., the developer could have been informed about the issue when adding vulnerable code to the project. On average, the comment was more than a year (495 days) present on Stack Overflow at the time of commit to GitHub.

Overall, we found that it takes, on average, almost four years (1,394 days) for a security-relevant comment to appear and, on average, nearly ten months (296 days) for a revision that fixes the security issue. However, we found three posts whose fixes occurred on the same day an issue was raised, and these posts affected three projects, with one of the posts (ID 779960) containing three issues that were commented on and fixed on the same day.

Discussions on Stack Overflow contribute to the evolution of code on the platform. When those discussions have a security nature, they trigger fixes to bugs and vulnerabilities in code snippets. These findings show that prior works' treatment of Stack Overflow code as

TABLE 5: Descriptive statistics for each event type measured in days. Margin of error for 95% confidence.

Events (in days)		Min	Median	Max	Average
All events ($N = 29$)					
t_{posted}	$\rightarrow t_{comment}$	270	1,394	3,639	$1{,}394 \pm 338$
$t_{comment}$	$\rightarrow t_{revision}$	0	20	1,516	296 ± 200
Commit before comment ($N = 17$)					
t_{posted}	$\rightarrow t_{clone\_before}$	89	336	2,289	556 ± 251
t_{clone_before}	$\rightarrow t_{comment}$	20	815	3,303	$1{,}060 \pm 506$
Commit after comment ($N = 12$)					
$t_{comment}$	$\rightarrow t_{clone\_after}$	9	224	1,221	495 ± 236

static limits the ability to build software tools to aid developers in reusing code from Stack Overflow more securely. We believe that if open-source contributors are equipped with a tool that consistently monitors Stack Overflow for security warnings and code fixes to reused code, those discussions and fixes can be brought timely to their notice. This will allow for fixes by the community to propagate faster to developer code bases.

4.3. Responsible Disclosure to Maintainers

We have disclosed the bugs and vulnerabilities we found to the maintainers of the affected projects. We used the dedicated security mailing lists to disclose the security issues found in Apache and Eclipse projects. For projects that do not define a specific policy, we directly emailed the main project contributor (i.e., the one with the most commits). We opened a public GitHub issue ticket for projects with only bugs (i.e., no security ramifications). We sent out 51 notifications, one for each bug/security issue we found, to the maintainers of the 43 affected projects. We received 40 responses (from 36 projects), with three indicating the project is no longer maintained and will not be updated. The remaining 33 projects responded and provided a fix while the maintainers of seven projects did not respond.

5. Discussion and Limitations

Similar to the security implications that outdated library versions pose on the security of production systems [24], [25], [39], outdated code snippets reused from Stack Overflow can have security implications for open-source projects. In our results, half of the code snippets reused in open-source projects were outdated (RQ1). We did not find evidence that developers updated the code snippets reused from Stack Overflow after the snippets evolved to newer versions (RQ2). While prior work has shown that developers reused vulnerable code snippets from Stack Overflow [3], [6], [9], [12], [26], our results additionally show that such vulnerabilities on Stack Overflow are being discussed and pointed out by the Stack Overflow community, which may consequently lead to a fix on Stack Overflow. However, those fixes did not propagate to the analyzed open-source projects on GitHub (RQ3). Looking at developers that attributed Stack Overflow as the source for copied code snippets, our results show that they are not immune to this problem: 13 (30%) of the 43 affected GitHub projects contain attributions to 6 (23% out of 26) Stack Overflow posts. However, despite attribution, developers were unaware that the reused snippets received security discussions and code fixes.

Towards better tool support: Our results provide the security research community with new insights into the tool support that can help developers avoid reusing insecure Stack Overflow code snippets. While software projects have tools (e.g., Dependabots) for managing library dependencies and keeping them up to date, there are no tools for managing dependencies on Stack Overflow snippets. Tools by prior work [12], [13] treat code snippets on Stack Overflow as *static* and, as a result, do not address the problem of "evolving" code snippets that need to be monitored for changes.

Based on our results, we envision providing developers with tools that make it seamless for them to *monitor updates* on Stack Overflow. For example, Visual Studio Code (VS Code) extensions [40], [41] provide a Stack Overflow search inside the IDE. However, these tools are limited to only searching *SO* for code snippets. Instead, they should support monitoring the Stack Overflow posts from which a developer copied code and inform developers about snippet updates and security discussions. This could be based on machine learning and natural language inference as in *Hark* [42] to detect security-relevant discussions and filter the large fraction of inconsequential code changes [34]. We believe this tool support will ultimately bring us closer to solving the problem of insecure code reuse.

5.1. Security Classification

A big limitation in general for studying Stack Overflow and our work is the lack of tools/methods for detecting vulnerable code snippets *in general*. Dicos [9] is a good step in the right direction. However, as we discovered, it is still too narrow in its language support and scope of vulnerabilities to be suitable as an automated classifier for our initially large data set. Consequently, in our methodology, we reduced the problem space by focusing on *outdated* and *filtered* code snippets, whose total number is feasible for manual verification. Thus, our classification trades large scaling for higher reliability and versatility. Due to this necessary trade-off, our methodology might miss security-relevant discussions in other threads on Stack Overflow that were filtered out. For example, in our study, the filter pipeline discarded 569 *potentially* security-relevant posts reused in 9,971 file sections (11.79% of 84,542). For 4,335 of those file sections, the pipeline selected another security-relevant post as the best candidate. For the remaining 5,636 file sections, the filter pipeline selected a *non*-security-relevant post and filtered out 327 distinct potentially security-relevant posts. We manually confirmed 202 of the 327 posts as not security-relevant, i.e., the pipeline did not miss a security issue discussed in those Stack Overflow posts. The filter pipeline selected 9 (of the remaining 125) true positive security-relevant posts as the best candidate for at least one other file section but always

favored a non-security-relevant post over 116 of the security-relevant posts reused in 1,112 (1.3%) file sections, for which we hence *may* have missed a security issue. Posthoc analysis of our filtering process indicates that the filter pipeline tends to favor non-security-relevant posts over security-relevant posts ((χ^2, df=1, N=6407) = 11.0674, p <0.001; see Appendix B), which might lead to under-reporting the number of projects with missing security fixes. However, the corresponding effect size (Cramer's V = 0.042) can be considered negligible and should have very little impact on our results. Thus, a refined approach without a filter pipeline might find more GitHub projects with outdated, insecure code snippets from Stack Overflow than the 43 projects we discovered. Still, our results demonstrate that the problem of *missing security fixes* exists and requires further attention.

The security filter identified 505 posts with security-relevant keywords (see §4.2.3), of which 238 were manually confirmed true positives and 267 confirmed false positives. This means the security filter has a ≈50% chance of correctly classifying a post as security-relevant. In addition, we manually verified the remaining 1,785 outdated posts not matching the keywords of the security filter and found four with actual security-relevant edits (i.e., 0.22% false negatives). This means the security filter misses very few but over-reports security-relevant posts, necessitating additional manual verification work of true positives. Moreover, like prior work [9], our context-based approach to determine security and bug fixes in Stack Overflow change histories is limited because commit messages on Stack Overflow are optional, and we relied on the quality of discussions by Stack Overflow users. Consequently, when there are no discussions about insecurities or commit messages mentioning a fix, our approach considers those cases secure by default.

Lastly, to ensure a high external validity of our results and minimal false positives, we used a conservative approach to select the most probable code snippet clone in our filter pipeline, fine-tuned and selected the minimum similarity threshold for NiCad, and relied on the fine-tuned parameters from Baltes et al. [11] to detect exact clones with PMD CPD. As a result, differences in the false negative rates of both clone detectors might lead to underestimating the number of outdated file sections for some of the programming languages more than for others.

Despite the trade-off in data set size and the limitations in clone detection and snippet classification, we discovered several popular GitHub projects in different programming languages affected by various security issues in outdated reused code. Still, those results should be seen as a lower bound to the problem of insecure, reused, and outdated Stack Overflow snippets on GitHub.

5.2. Origins of Common Code

In this work, we studied how code reused between Stack Overflow and GitHub evolves. To create a feasible data set of snippets for our study, the methodology in this paper assumes code provenance from Stack Overflow posts to GitHub projects, i.e., that GitHub developers copy code from Stack Overflow. Hence, when a code fragment appears on Stack Overflow and GitHub, we focused on code snippets that appear *first* on Stack Overflow before their GitHub counterparts. However, a code snippet on both GitHub and Stack Overflow could have been independently copied from a third-party source. For example, prior work [1], [8], [23], [43] has shown that some code snippets on Stack Overflow are copied over from tutorials, from GitHub projects, or other software collections (e.g., Qualitas). The 43 affected GitHub projects in our results did not contain an attribution to external sources other than Stack Overflow.

Considering the rich discussions on Stack Overflow, which we also leveraged for our security classification of outdated snippets, we think it very promising for future work to bridge the gap between these two knowledge-sharing ecosystems [8] and support developers with insights from Stack Overflow whenever they use code that is present on Stack Overflow *irrespective of whether the code was copied from Stack Overflow* or not. This could further enhance tool support for developers, as discussed earlier. For our current security-oriented study, this was impossible considering the sheer volume of this code overlap (see §4.1.2), and we focused on outdated snippets that might have been copied from Stack Overflow to GitHub.

6. Conclusion

We measured the impact of Stack Overflow code artifact evolution on software projects that rely on specific versions of code snippets reused from Stack Overflow. The presence of duplicate code on Stack Overflow made it non-trivial to pinpoint an exact snippet version reused in a section of a source file in an open-source project. This prompted us to build a filter pipeline to reduce the number of code snippet versions reported as clones in a single source file section. By analyzing the timeline of reused code snippets, we found that 51% of code snippets reused from Stack Overflow were outdated, which affected 2,109 open-source projects. At the same time, we found no evidence that developers update reused versions of Stack Overflow code snippets, regardless of whether the snippets underwent bug or security fixes on Stack Overflow. Finally, by examining comments and commit messages of Stack Overflow posts, we found fixes to 15 security issues that are missing in 43 GitHub projects, with an additional 20 general code improvements missing in 26 GitHub projects. Considering the popularity of the 43 projects missing security fixes, we believe this number to be the lower bound of the affected code bases. Our findings call for action to provide developers with tools to track Stack Overflow long-term for security-focused discussions and code fixes to reused code.

Availability. Our data set and tools are available at https://osf.io/s2vgm/?view_only=785ada7b1efd4ac6aaf5a77cc5123076.

References

[1] 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.

[2] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *Proc. 28th International Conference on Software Maintenance (ICSM)*. IEEE Computer Society, 2012.

[3] 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)*. IEEE Computer Society, 2016.

[4] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: An analysis of android ssl (in)security," in *Proc. 19th ACM Conference on Computer and Communication Security (CCS '12)*. ACM, 2012.

[5] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking ssl development in an appified world," in *Proc. 20th ACM Conference on Computer and Communication Security (CCS '13)*. ACM, 2013.

[6] 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.

[7] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *Proc. Cybersecurity Development (SecDev'17)*. IEEE Computer Society, 2017.

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

[9] 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.

[10] 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, pp. 2359–2375, jul 2022.

[11] 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.

[12] 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' 19)*. USENIX Association, 2019.

[13] 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.

[14] Stack Overflow. (2022) How to pretty print xml from java? [Online]. Available: https://stackoverflow.com/a/1264912/8462878

[15] 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.

[16] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack overflow in github: Any snippets there?" in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017.

[17] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *2009 IEEE 31st International Conference on Software Engineering*, 2009.

[18] M. Gharehyazie, B. Ray, and V. Filkov, "Some from here, some from there: Cross-project code reuse in github," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017.

[19] T. Diamantopoulos, M.-I. Sifaki, and A. L. Symeonidis, "Towards mining answer edits to extract evolution patterns in stack overflow," in *Proc. 16th International Conference on Mining Software Repositories (MSR '19)*. IEEE Press, 2019.

[20] M. Ahmad and M. O. Cinnéide, "Impact of stack overflow code snippets on software cohesion: A preliminary study," in *Proc. 16th International Conference on Mining Software Repositories (MSR'19)*. IEEE Press, 2019.

[21] 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.

[22] S. Baltes and C. Treude, "Code duplication on stack overflow," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 13–16. [Online]. Available: https://doi.org/10.1145/3377816.3381744

[23] M. A. Nishi, A. Ciborowska, and K. Damevski, "Characterizing duplicate code snippets between stack overflow and tutorials," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19. IEEE Press, 2019, p. 240–244. [Online]. Available: https://doi.org/10.1109/MSR.2019.00048

[24] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 356–367. [Online]. Available: https://doi.org/10.1145/2976749.2978333

[25] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web," in *Network and Distributed System Security Symposium (NDSS)*, 2 2017.

[26] R. Abdalkareem, E. Shihab, and J. Rilling, "On code reuse from stackoverflow: An exploratory study on android apps," *Information and Software Technology*, vol. 88, pp. 148–158, 2017.

[27] Stack Overflow. (2023) Stack overflow trends. [Online]. Available: https://insights.stackoverflow.com/trends

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

[29] G. Gousios, "The ghtorent dataset and tool suite," in *Proc. 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, 2013.

[30] C. K. Roy and J. R. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *Proc. 16th IEEE International Conference on Program Comprehension*, 2008.

[31] PMD Source Code Analyzer Project. (2022) Finding duplicated code with cpd. [Online]. Available: https://pmd.github.io/latest/pmd_userdocs_cpd.html

[32] Stack Overflow. (2022) How to make a copy of a file in android? [Online]. Available: https://stackoverflow.com/a/9293885/8462878

[33] ——. (2022) Standard concise way to copy a file in java? [Online]. Available: https://stackoverflow.com/a/19542599/8462878

[34] S. Baltes and M. Wagner, "An annotated dataset of stack overflow post edits," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1923–1925. [Online]. Available: https://doi.org/10.1145/3377929.3398108

[35] K. Pan, S. Kim, and E. J. Whitehead, "Toward an understanding of bug fix patterns," *Empirical Softw. Engg.*, vol. 14, no. 3, p. 286–315, jun 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9077-5

[36] H. Osman, M. Lungu, and O. Nierstrasz, "Mining frequent bug-fix code changes," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, 2014, pp. 343–347.

[37] Common Weakness Enumeration. (2022) Common weakness enumeration. [Online]. Available: https://cwe.mitre.org/

[38] Stack Overflow. (2022) How do i trim leading/trailing whitespace in a standard way? [Online]. Available: https://stackoverflow.com/a/122721/8462878

[39] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on android," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2187–2200. [Online]. Available: https://doi.org/10.1145/3133956.3134059

[40] Extensions for Visual Studio Code. (2023) Stackoverflow instant search. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=Alexey-Strakh.stackoverflow-search

[41] ——. (2023) Stackfinder. [Online]. Available: https://marketplace.visualstudio.com/items?itemName=mark-fobert.stackfinder

[42] 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 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2469–2486. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833729

[43] 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, pp. 560–581, 2021.

Appendix A. Calibrating NiCad clone detector

We needed to find the optimal NiCad parameters for working with Stack Overflow code snippets. To this end, we calibrated NiCad's *threshold* and *minsize* configuration parameters, i.e., the minimum number of LoC and the type of clone (type-1, type-2, or type-3).

To measure the performance of different configuration settings, we extracted $3,824$ posts that are *directly attributed* in the SOTorrent dataset as a ground truth. We removed all attributed question posts from this set for two reasons: 1) developers are more likely to copy code snippets of answer posts [11]; 2) we found that many developers tend to wrongly attribute the question of the answer that contains the snippet. Of the $3,824$ answers, 364 did not contain any code snippets. From the remaining $3,460$ answers, we extracted $4,999$ code snippets after filtering out all snippets with less than five lines of code—the minimum code length for NiCad—resulting in a total of $2,656$ code snippets. Figure 9 shows the distribution of the LoC of code snippets in the resulting attributed dataset. For each answer post in the dataset, we downloaded all the Java files on *GH* in which the corresponding answer was attributed. With this set of Stack Overflow posts and Java files, we measure the performance of our clone detection based on *precision* and *recall*.

TABLE 6: Effectiveness of different NiCad configurations in detecting attributed code snippets in GitHub source files.

Configuration			Clone Detection				
$minsize$	C_{SO}	type	C_{nicad}	TP	FP	Prec	Rec
5	2,656	$type1$	317	317	0	100%	12%
		$type2$	374	368	6	98%	14%
		$type3$	667	646	21	97%	24%
10	770	$type1$	196	196	0	100%	26%
		$type2$	227	227	0	100%	30%
		$type3$	473	464	9	98%	60%
15	452	$type1$	108	108	0	100%	24%
		$type2$	126	126	0	100%	28%
		$type3$	297	295	2	99%	65%

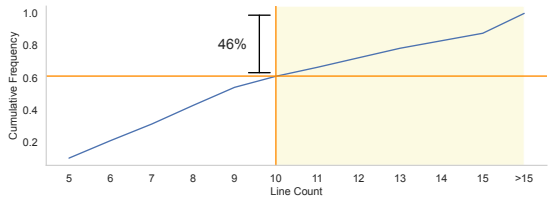


Figure 9: Distribution of the LoC for code snippets in the attributed dataset. The shaded area indicates the snippet sizes our NiCad configuration covers.

Table 6 summarizes the results of running NiCad for the different configuration parameters, where C_{so} is the set of snippet clones attributed in Java files and C_{nicad} is the set of snippet clones identified by NiCad. We manually verified all reported clones and recorded each run's true and false positives. While a $minsize = 5$ would theoretically be able to detect all $2,6565$ clones, we found that NiCad has a very low recall for this parameter. This is rooted in a conservative classification of true positives: in most cases, it was very hard to tell, with so few code lines, whether the snippet was reused from Stack Overflow or the developer actually wrote the snippet. A $minsize = 15$ does not work well either since the number of potentially detectable clones is very small ($C_{SO} = 452$ or 17% of all possible clones). The reason is that code snippets on Stack Overflow are, on average, 12 lines long [1], and setting a minimum threshold of 15 lines will exclude many snippets. Thus, we selected $minsize = 10$, where the best balance between precision and recall is achieved when detecting type-3 clones. We suspect the reason to be that most code snippets on Stack Overflow are not compilable, and the import statements for libraries are rarely included in the snippets, necessitating code adaptions of type-3. Thus, we selected $< (minsize = 10, type3 >$ as final calibration for NiCad.

We use the selected parameters and ran the clone detector on the Java and C code snippets sample shown in Table 1. Figure 10 shows the cumulative frequently distribution for the NiCad similarity values.The vast majority (80,665 or 95.4%) of the clones are identical (NiCad score of 100). To also include highly similar but not identical clones, we need to determine a cutoff point for the similarity score. We manually inspected some of the reported clones for each

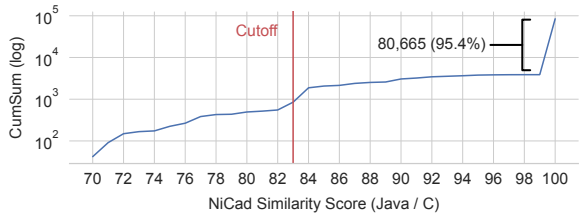


Figure 10: Cumulative sum of detected clone pairs for NiCad similarities above 70%.

TABLE 7: Cross table between posts **filtered** vs. **flagged** by our security filter, with the observed values and the expected values.

	Not Flagged	Flagged	Total
Not Filtered Out	3,267 (3213.02)	974 (1027.98)	4,241
Filtered Out	1,587 (1640.98)	579 (525.02)	2,166
Total	**4,854**	**1,553**	**6,407**

language and noticed that some *Java* clone pairs with a 70% similarity score looked structurally similar but semantically different. Since this may skew our results, we resorted to manually verifying a random sample of the reported clones and selected clones belonging to five similarity groups (i.e., 70%, 75%, 80% and 85%). The aim of the manual verification process was to select an optimal *similarity threshold* with which we can be confident that a detected clone is a true clone. The random sample consisted of 30 clone pairs for each similarity group—split equally among question and answer posts—and two researchers manually verified the clone pairs for each group. We found that 21 of the clone pairs in the 70% similarity group were false positives, 13 clone pairs in the 75% similarity group were false positives, 8 clone pairs in the 80% similarity group were false positives and 1 clone pair in the 85% similarity group was a false positive. We decided to additionally select a random sample of clone pairs in the 83% similarity group, manually verified those, and found 2 false positive clones. As a result, we selected 83% as a similarity threshold for clones of *Java* code snippets. We performed the same experiment for the reported clone pairs for the *C* language but found no false positives for the 70% similarity group. Thus we selected 70% as a similarity threshold for clones of *C* code snippets.

Appendix B.
Chi-Square Statistical Test

We conducted a Chi-Square (χ^2) test to understand whether the filter pipeline favors non-security-relevant posts over potentially security-relevant posts. Table 7 shows the observed and expected values based on the number of posts that are input to filter F3 and the output of filter F7. We excluded filters F1 (Attribution) and F2 (Commit Date) to focus on potential security-relevant posts discarded by the heuristics-based part of the filter pipeline.

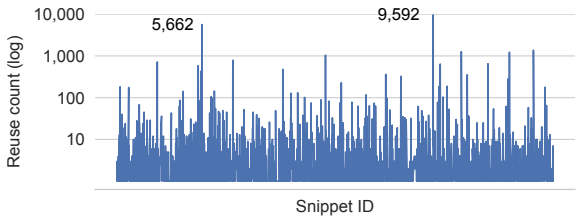


Figure 11: Frequency of reused individual snippets.

TABLE 8: Top-10 most reused outdated snippets

Snippet ID	Reuse count	Snippet ID	Reuse count
184588891	9,592	121742444	1,044
49722853	5,662	67677724	788
243061964	1,372	23640102	717
200993642	1,271	216546726	651
229059551	1,227	188694515	640

Appendix C.
Frequency of Outdated Snippet Reuse

Figure 11 depicts how often each of the 2,302 distinct snippets were reused in an outdated version in GitHub projects. The average outdated snippet in our data set is reused 16.87±9.83 times. Table 8 lists the top-10 most reused outdated snippets in our data set.

Appendix D.
Projects Missing Improvements

Table 9 shows the list of GitHub projects missing various improvements to code reused from Stack Overflow. Java projects (17/20) were mainly affected by those and were concerned with the release of Java 8 in which functional APIs were added to the standard library. Together, these projects were watched on average 179.2 times (max. 783), received an average of 3,790.9 stars (max. 12,361), and were forked on average 94.6 times (max. 809).

Appendix E.
Projects Missing Security Fixes

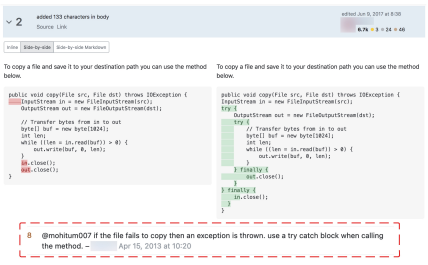
Table 10 shows the list of GitHub projects missing security fixes on Stack Overflow.

Appendix F.
Additional Examples of Insecure Snippets

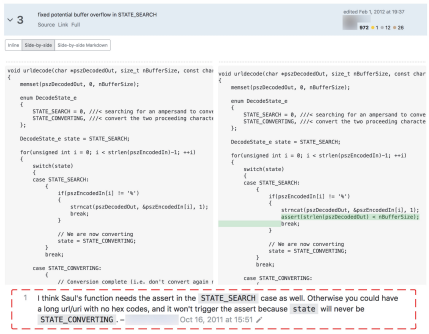
In Section 2, we provided an example showing how a fix to an *XML eXternal Entities (XXE)* injection vulnerability was missed in the Apache Lucene-Solr and the Apache Chemistry projects. Here, we provide three additional examples of other issues we found.

TABLE 9: GitHub projects missing fixes to code improvements on Stack Overflow

Project	Language	Watch	Fork	Star	Contributors	Last Commit (year)
OsmAnd (OSM Automated Navigation Directions)	Java	138	875	3,115	809	2022
Azure SDK for Java	Java	280	1,431	3,998	447	2022
F-Droid Client	Java	57	104	706	353	2022
GeoTools (The Open Source Java GIS Toolkit)	Java	110	1,008	1,274	235	2022
J2ObjC (Java to Objective-C Translator and Runtime)	Java	307	920	5,877	82	2022
sshj (SSHv2 library for Java)	Java	118	517	2,094	69	2022
Eclispe Deeplearning4J (DL4J)	Java	783	4,923	12,361	57	2022
MGit (Git client for Android)	Java	45	121	594	56	2022
GassistPi (Google Assistant for Single Board Computers)	Python	77	309	977	16	2022
Apache POI	Java	76	593	1,401	15	2022
PictureSelector	Java	199	2,635	11,450	5	2022
WebRTC Chrome Extensions	JavaScript	80	483	894	5	2022
Open Event Website App Generator	JavaScript	38	870	1,990	89	2021
React Designer	JavaScript	42	241	1,794	8	2021
LitePal for Android	Java	293	1,587	7,835	4	2021
w3af (Web Application Attack and Audit Framework)	Python	193	1,151	3,917	64	2020
WebRTC-Experiment	JavaScript	669	3,850	10,603	11	2020
Matisse	Java	241	2,011	12,258	28	2019
MusicDNA	Java	93	590	2,737	8	2019
libstreaming	Java	267	1,036	3,215	6	2019
Friend Spell	Java	11	74	415	6	2017
Pretty Curved Privacy (PCP)	C	7	5	125	3	2017
InstaMaterial	Java	307	1,498	5,028	3	2016
VOMS Java API	Java	10	7	5	4	2016
Livestreamer	Python	208	617	3,844	75	2016
Breeze	C	9	18	57	1	2013
Average		**179.2**	**1056.7**	**3790.9**	**94.6**	
Max		**783**	**4,923**	**12,361**	**809**	



(a) Answer 9293885 with CWE-404.



(b) Buffer Overflow issue fixed in version 3 of the post.

(c) The most reused, outdated, buggy (no CWE) snippet with a subsequent fix.

Figure 12: Examples of insecure snippets.

Improper Release of Resource (CWE-404). This issue was found and fixed in 3 distinct posts and the versions containing the issue are reused in 6 *GH* projects. Figure 12a shows two versions of post #9293885: The version on the left copies data from a source input stream to a destination output stream. Should an exception be thrown during the copy operation, the closing API call will not be reached and the JVM will not garbage collect the resource.

This post is the accepted answer in the thread and has at the time of writing three versions. The insecure version was introduced in version #1 of the post on Feb 15, 2012 and 425 days later, on April 15, 2013, a comment was made indicating that the file resource may not be properly garbage collected when an exception is thrown. Three comments raise this issue, and we pick the earliest comment of those. The issue was subsequently fixed on June 9, 2017, by the author of the answer in version # 2 of the post. However, the fix did not propagate to the 6 projects containing the insecure version.

Buffer Overflow (CWE-120). Answer post 2766963 (see Figure 12b) contains a buffer overflow vulnerability that was introduced in version #1 of the answer on May 4, 2010. The vulnerable version was reused in the *STATSD-C* project and committed on Oct. 25, 2011. On Oct 16, 2011 (530 days after it appeared on Stack Overflow), a comment raised the issue, which was subsequently fixed by the same author in version #3 of the post on Feb 1, 2012.

TABLE 10: GitHub projects missing bug/security fixes to code on Stack Overflow. Two projects (*) were retired months after we curated our GitHub projects sample; nonetheless, the maintainers were still notified.

Project	Weakness	Language	Watch	Fork	Star	Contributors	Last Commit (year)
Odoo	Undefined Behaviour	JavaScript	1,494	16,121	24,881	1,385	2022
Wikimedia Commons Android app	CWE-404	Java	60	968	731	270	2022
The Fuck	Others	Python	848	3,189	70,614	176	2022
Apache NetBeans	CWE-404	Java	157	689	1,836	175	2022
Amazon S3cmd	Others	Python	103	850	3,962	175	2022
Open Event Frontend	CWE-1339	Python	22	1,694	2,188	151	2022
CARTO	CWE-690	JavaScript	207	672	2,582	134	2022
logback	CWE-404	Java	167	1,115	2,404	104	2022
Cider	CWE-754 (2)	JavaScript	24	174	348	65	2022
Apache Nutch	Undefined Behaviour	Java	240	1,199	2,356	46	2022
JPEXS Free Flash Decompiler	Others	Java	187	534	3,128	33	2022
Python Audio Analysis Library	CWE-754	Python	207	1,083	4,742	22	2022
WordOps	CWE-772	Python	58	167	866	21	2022
Eclipse N4JS	CWE-772	Java	11	25	26	18	2022
Weevely	Undefined Behaviour	Python	132	548	2,596	18	2022
Gmail Backup Software (Gmvault)	CWE-172	Python	80	272	3,429	16	2022
RomRaider	CWE-172	Java	80	272	3,429	15	2022
Kubebox	CWE-690	JavaScript	47	139	1,912	13	2022
*AndroidTreeView	CWE-475	JavaScript	84	614	2,884	5	2022
Chrome-Extensions	CWE-1339	JavaScript	80	483	894	5	2022
Apache Lucene-Solr	CWE-611	Java	315	2,723	4,357	234	2021
MoneyManagerEx for Android	CWE-404	Java	49	167	327	17	2021
Apache Fineract Android Client	CWE-404	Java	20	147	31	13	2021
Faster-RCNN in Tensorflow	Others	Python	88	1,149	2,461	7	2021
STATSD-C	CWE-120	C	6	13	75	6	2021
Faster RCNN with PyTorch	Others	Python	52	464	1,609	3	2021
Gnucash for Android	CWE-475	Java	101	525	1,143	46	2020
AppScale GTS	Undefined Behaviour	Python	159	293	2,419	39	2020
Eclipse Ceylon	Undefined Behaviour	Java	41	64	385	34	2020
HowManyPeopleAreAround	Others	Python	165	383	6,687	12	2020
WebRTC-Experiment	CWE-1339(2)	JavaScript	669	3,852	10,604	11	2020
*Apache Chemistry	CWE-611	Java	11	60	46	6	2019
Chinese OCR	Others	Python	93	1,077	2524	1	2019
shadowsocks-libev	Others	C	14	900	87	81	2017
OpenPGP for Android	Others	Java	31	79	237	59	2017
Mirai BotNet	CWE-20, CWE-704	C	545	3,362	7,323	5	2017
FastER RCNN (TFFRCNN)	Others	Python	52	431	890	1	2017
ngrok-libev	CWE-754, CWE-835, CWE-704	C	9	25	34	1	2016
Layout Cast	Others	Python	71	185	1,711	5	2015
Hashkill Password Recovery Tool	CWE-20, CWE-754, CWE-835	C	23	55	183	3	2014
Breeze (Simple) HTTP Server	CWE-194	C	9	19	57	1	2013
PHP bindings to libsass (sassphp)	CWE-704	C	2	67	40	1	2012
Tigger	CWE-704	C	2	6	40	6	2011
Average			**157.4**	**1,085.4**	**4089.6**	**79.7**	
Max			**1,494**	**16,121**	**70,614**	**1,385**	

Most Buggy Snippet (No CWE). Figure 12c shows the most reused, outdated, buggy (no CWE assigned) code snippet from answer `377028`. The insecure version #1 of the answer was posted on December 18, 2008, and reused in 8 open-source projects. A comment on July 22, 2011 (946 days after posting) pointed out that the code might incorrectly match directories with the *+x* bit set. The snippet was fixed 260 days later; however, the fix did not propagate to the 8 *GH* projects.

Appendix G. Meta-Review

G.1. Summary

This paper presents a measurement of code similarity between Stack Overflow and Github to determine whether developers change code copied/pasted from Stack Overflow if the post is fixed after the initial copy to remove a vulnerability. The paper presents a pipeline using clone detection and commit/comment-based verification to identify and confirm security-relevant edits. The method is evaluated on 1.5M snippets and 11,479 projects, revealing that 2,109 projects use outdated code and 43 projects have unpatched security issues from Stack Overflow snippets.

G.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field

- Creates a New Tool to Enable Future Science
- Identifies an Impactful Vulnerability

G.3. Reasons for Acceptance

1) The paper provides a valuable step forward in an established field. The paper investigates whether developers track code copied from StackOverflow that may include vulnerabilities to ensure their code is appropriately fixed when those edits are made on StackOverflow and demonstrate the security impacts associated with this lack of tracking.
2) The paper creates a new tool to enable future science. This paper created a pipeline for comparing code on StackOverflow and Github to identify vulnerabilities based on code fixes in StackOverflow.
3) This paper identifies an impactful vulnerability. The paper studies the vulnerabilities introduced by open-source contributors' failure to track Stack Overflow (SO) code snippet security updates. By taking advantage SOTorrent dataset and crawling open-source Github repos, the paper evaluates 1.5M code snippets over 11,479 projects, finding 2109 projects using outdated code and 43 projects having security issues.

G.4. Noteworthy Concerns

There remain some questions about the accuracy of the filtering pipeline. Two reviewers believed there may still be inaccuracies in the filtering in cases where known attribution is not available and where false-negative cases due to clone detection errors (specifically with PMD CPD) occur.