On the Informativeness of Security Commit Messages: A Large-scale Replication Study

Syful Islam
syful.islam@telecom-paris.fr
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

Stefano Zacchiroli
stefano.zacchiroli@telecom-paris.fr
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

Abstract

The informativeness of security-related commit messages is crucial for patch triage: when high, it enables the rapid distribution and deployment of security fixes. Prior research (Reis et al., 2023) reported, however, that commit messages are often too uninformative to support these activities. To assess the robustness of this negative result, we independently replicate the original study using only the information provided in the paper, without reusing any of the original artifacts (data, analysis pipeline, etc.).

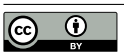
Unlike the original study, we source commit data not only from GitHub, but from the entire Software Heritage archive, which includes projects hosted on many other platforms and using multiple version control systems. We retrieve 50 673 security-related commits and analyze their informativeness using an independent re-implementation of the techniques introduced by Reis et al. For the same source (i.e., GitHub) and time period (from June 1999 to August 2022) as the original study, our replication confirms the original findings in a statistically significant way: security-related commit messages are, in general, not informative enough for security-focused purposes.

We then extend the original study in several ways. Over a longer time period (from June 1999 to October 2025), we find that commit-message informativeness is worsening. Breaking results down by software ecosystem (Linux kernel, Ubuntu, Go, PyPI, etc.), we observe significant differences in informativeness. Finally, we examine emerging best practices for writing commit messages, such as the Conventional Commits Specification (CCS), and again find significant differences in an unexpected direction: CCS-compliant commits are less informative than non-compliant ones.

Our findings highlight the need for cross-ecosystem analyses to understand platform- and community-specific commit-message practices, and to inform the development and adoption of universally applicable guidelines for writing informative security-related commit messages.

CCS Concepts

• **Software and its Engineering → Security and Privacy**; • **NLP → Software Artifacts**.



EASE 2026, Glasgow, Scotland, United Kingdom

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Keywords

commit messages, patch management, software security, replication study, conventional commits

ACM Reference Format:
Syful Islam and Stefano Zacchiroli. 2026. On the Informativeness of Security Commit Messages: A Large-scale Replication Study. In *Proceedings of The 30th International Conference on Evaluation and Assessment in Software Engineering (EASE 2026).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

In today's rapidly shifting cybersecurity landscape, software applications are under constant threat of security attacks from a variety of sources and are therefore viewed as a matter of great concern [9]. While many efforts have been made to prevent security attacks, they are still growing and eventually pose significant threats to IT industries worldwide.

To mitigate security attacks, developers generally propose code changes (i.e., patches) along with necessary descriptions via commit messages to the software repository, which are afterward analyzed by the maintainers. In some cases, maintainers are hesitant to deploy updates because the proposed change is not adequately documented as security-relevant and is therefore not deemed critical [32]. Such consequences can lead to serious security attacks on target software applications. For instance, the Equifax security breach occurred due to failure of patching a bug in Apache Struts (i.e., CVE-2017-5638) and exposed sensitive data of 143 million US consumers, yet a patch for that vulnerability had been available for months before the compromise occurred [10, 16]. Hence, timely security patching is an essential safeguard and one of the most effective and widely used strategy to prevent cyberattacks on software applications [19].

Security patch triage management is a structured process to evaluate, prioritize, and manage security patches before they are deployed and is therefore crucial in the contemporary software development life cycle [13]. However, effective patch triage management is ultimately dependent on the description of changes via commit messages or the availability of references in public vulnerability databases [28]. Several previous studies reported that well-structured and informative commit messages that explicitly describe security implications can facilitate automated vulnerability detection and human understanding [14, 24]. In contrast, other studies reported that commit messages often fail to provide sufficient security-related details and lack explicit references to security issues, limiting their utility for vulnerability assessment [20, 21]. Specifically, Reis et al. [25] reported that security commit messages are not informative enough for prototyping an effective patch triage management system.

To assess the robustness of this negative result, we independently replicate the original study using only the information provided in the paper [25], without reusing any of the original artifacts (data, analysis pipeline, etc.). In detail, we perform a large-scale replication study on a 50 673 security commit message dataset (from 24 June 1999 to 07 October 2025) sourced from Software Heritage to assess security commit message informativeness and whether content richness differs significantly by software ecosystem and usage of emerging guidelines for writing structured commit messages such as Conventional Commit Specification (CCS) [2]. We used the well-known named entity recognition (NER) technique along with a pretrained spaCy model and entity dictionary, as the original work did. Finally, we classified the information level (i.e., Excellent, Very Good, Good, Medium, Poor, and Very Poor) of security commit messages based on entity categories and predefined rules accordingly. Specifically, our work investigates the following research questions:

- *RQ_1 To what extent can we replicate the results of Reis et al. [25] about the informativeness of security-related commit messages?*
  For the same source (i.e., GitHub) and time period (from June 1999 to August 2022) as the original study, our replication confirms the original findings in a statistically significant way: security-related commit messages are, in general, not informative enough for security-focused purposes.
- *RQ_2 Does security commit message informativeness remain consistent over temporal scope and other code-hosting platforms?*
  Over a longer time period (from June 1999 to October 2025), we find that commit message informativeness is worsening. In addition, we find that security-related commit messages of other code-hosting platforms (e.g., GitLab, git.kernel.org, etc.) are better in term of content richness compared to GitHub.
- *RQ_3 Does security commit message informativeness differ significantly by software ecosystems?*
  Security commit message informativeness differs significantly across software ecosystems, highlighting ecosystem context as an important factor in determining the documentation quality of security-related changes.
  Overall, security commit messages from operating system-related software ecosystems (e.g., Android, Linux, Ubuntu, etc.) are generally more informative than those from other ecosystems.
- *RQ_4 Does security commit message informativeness differ significantly among commits that comply with the Conventional Commit guidelines and others that do not?*
  Perhaps surprisingly, we find that commits that adhere to emerging guidelines for writing commit messages like the Conventional Commit Specification (CCS) are *less informative* than non-compliant ones.
  CCS guidelines alone appear insufficient to raise information levels although they intended to standardize and improve commit messages.

Overall, these findings highlight the necessity of conducting cross-ecosystem studies to better understand differences in security maintenance practices followed by stakeholders and to support the development of standard guidelines applicable to all ecosystems for writing informative security commit messages, thereby improving automated patch triage management systems. In addition, software ecosystem communities should reevaluate their existing security policies and approaches to better motivate stakeholders to implement best practices for documenting security-related changes. We believe that enriching security commit messages and maintaining this knowledge base should become a coordinated industry-wide effort, bringing benefits for IT organizations in the rapidly shifting cybersecurity landscape.

The rest of the paper is organized as follows. Section 2 describes our replication methodology. Section 3 reports our empirical study results. Section 4 discusses the implications and recommendations of our findings. Section 5 discusses the threats to validity, and Section 6 concludes the paper.

Data availability: see dedicated section at the end of the paper.

2 Replication methodology

The main goal of this study is to examine the informativeness of security-related commit messages, by replicating and extending the previous study by Reis et al [25]. We hence also follow a methodology similar to theirs, as shown in Figure 1.

Replication terminology: In our replication of the original work we did not reuse any of the artifacts from the original paper—which does not come with a reproducibility package—data, code, etc. What we present in the following is hence a proper *replication* experiment of the original work ("Different team, different experimental setup", according to the ACM terminology [1]), rather than a reproduction ("Different team, same experimental setup").

2.1 Data collection and preprocessing

To build a security commit message dataset, we execute the following steps sequentially.

2.1.1 Step 1: Vulnerability metadata collection from public vulnerability databases. Initially, we downloaded the latest data dump versions available online from popular open-source vulnerability publishing websites, namely OSV [4] (Open-Source Vulnerability Database) and NVD [3] (National Vulnerability Database), as derived from original work. From OSV, we obtained a total of 557 427 unique vulnerability records up to 17 December 2025 for different software ecosystems such as Android, Debian, Go, Maven, and PyPI. From NVD, we obtained a total of 324 423 unique CVE records up to 27 December 2025. In total, we obtained 881 850 vulnerability records from both datasets. (As vulnerability reports can be duplicated across the two data sources, we later deduplicate vulnerabilities between them, as described later in this section.)

2.1.2 Step 2: Collection and preprocessing of references to security patches. Each vulnerability report from both databases (i.e., OSV and NVD) contains a "references" section that includes links to security patches when available.

Collection: To collect security vulnerability records with patch references, we initially excluded both OSV and NVD records that do not contain any reference. Afterward, we prepared a random sample from both datasets (i.e., 384 each) keeping 95% confidence with a 5% interval and manually analyzed the links to reveal patterns for

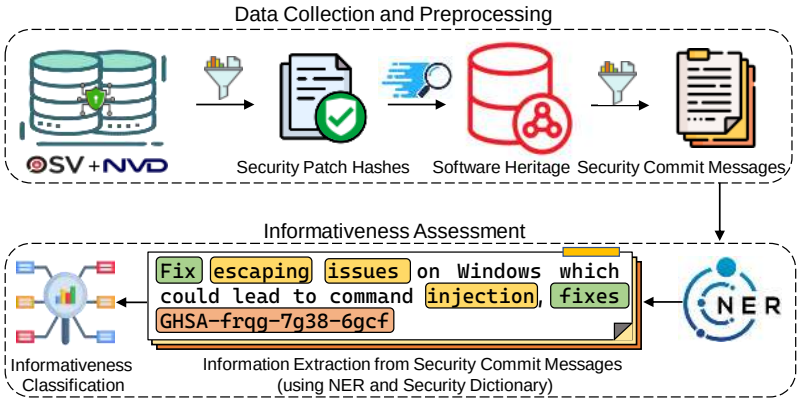


Figure 1: Overview of the methodology of our study.

accurate patch commit hash key (i.e., ID) extraction. After analyzing the sample records, we gathered and merged the patterns obtained from security patch links in both datasets. Next, we devised a regular expression based on the patterns and applied it to the links in the reference section of the vulnerability records. Thus, we obtain a total of 59 486 (10.67%) unique OSV records and 30 270 (9.33%) unique NVD vulnerability records containing at least one security patch link.

Deduplication and cleaning: Since many security vulnerabilities are recorded in both NVD and OSV datasets and mentioned through the aliases field, we found duplicate patch entries between both sources. To remove these duplicate entries and obtain unique commit hashes from vulnerability records, we executed the following operations. First, we merged both datasets by record identifiers after necessary normalization and obtained 65 535 unique vulnerability reports (90.77% from OSV and 9.23% from NVD). When deduplicating, we kept OSV records as they provide additional useful metadata, such as ecosystem information. Also, during deduplication, we merge missing information across the two databases; for example, if one of the two has a severity score missing from the other, we propagate it to the kept record.

Each vulnerability can be associated to multiple commits, in a one-to-many mapping, e.g., one report can contain multiple patch hashes such as GHSA-f5pm-c4cw-563p. Hence, we extracted and merged all commit hashes referenced from all vulnerability records into a single set, de-duplicating by commit hash. Thus, we ended up with a dataset of 84 435 unique security commit hashes, of which 94.07% are from OSV and 5.93% from the NVD dataset.

2.1.3 Step 3: Collection and preprocessing of security commit messages from the Software Heritage archive. To collect the message descriptions of patched commit hashes referenced in security vulnerability records, we utilize data from the Software Heritage archive [8], the largest archive of source code and its development history, as captured by various code-hosting platforms such as Git.

We executed the following steps to obtain the final security commit message dataset. (i) Before querying the Software Heritage graph dataset [23], we performed exploratory analysis on the security patch commit hash list to identify anomalies and noticed that some records included short versions of hashes, whereas the database natively stores each revision under the full hash (i.e., SHA-1 full-length key) due to its global uniqueness. In total, 581 commit hashes are short out of 84 435 security patch hashes. Since shortened commit hashes are not globally unique and may become ambiguous when data spans multiple repositories, leading to misidentification, we excluded short hashes with multiple returned messages to improve data accuracy and reduce false positive discovery from the Software Heritage database. Thus, we obtained a total of 80 695 security patch commit messages from 84 435 unique hashes, corresponding to the exclusion of 65 short hashes with multiple entries and 3675 hashes (both short and full) that no longer exist or were missed due to a slightly earlier available version of the Software Heritage database updated up to October 2025.

(ii) We found duplicated commit messages resulting from vulnerability records with references to the same fix applied in different branches. For example, two security commit hashes have duplicate messages because developers applied the same security fix (i.e., CVE-2015-0253) to different Apache httpd branches, so the patch was committed twice with identical descriptions as part of Apache's backporting workflow. In such cases, since the security commit messages are the same, we kept only one of them. Thus, an extra 29 367 commits were removed from the dataset, resulting in 51 328 security commit messages.

(iii) We then removed bot-generated commit messages and any pull request merges from Dependabot using non-human-written patterns and bot account names listed in previous works [7, 11, 12, 15, 31, 34]. This resulted in 50 978 human-written security commit messages and the removal of 350 bot/non-human-generated security commit messages.

(iv) We noticed that some of the commit messages were not written in English. As the subsequent step used to determine the informativeness of commit messages relies on English-specific natural language processing, non-English messages should be removed to avoid underestimation. Therefore, we ran the popular Python `langdetect` [6] package to infer text language and detected 3033 (5.95%) security commit messages not written in English. Considering that `langdetect` package can be inaccurate when evaluating too short or too ambiguous texts, the authors manually inspected all the automatically detected non-English messages to ensure English

Table 1: Different Categories of Entities [25].

Category	Rationale	Examples
VULNID	Identify vulnerabilities for different ecosystems in commit messages: CVE, GHSA, OSV, PyPI, etc. This ID can enable the detection of security commits.	GHSA-269q-hmxg-m83q, CVE-2016-2512, CVE-2015-8309, GHSA-9x4c-63pf-525f, OSV-2016-1
CWEID	Identify weakness type of the vulnerabilities and can enable assessment tasks. One common taxonomy used to classify security weaknesses is the Common Weakness Enumeration (CWE) one.	CWE-119, CWE-20, CWE-79, CWE-189
SEVERITY	Identify severity of discovered vulnerability through assigned score/level and can enable prioritization during patch management processes	low, medium, high, critical
SECWORD	Security-relevant words usually describe the vulnerability and respective fix. These words can enable the detection of security commits.	ldap injection, crlf injection, improper validation, command injection, cross-site scripting, sanitize, bypass
ACTION	A commit usually implies an action, in the case of security, fixing a vulnerability (corrective maintenance).	fix, patch, change, add, remove, found, protect, update, optimize, mitigate
FLAW	Fixing a security vulnerability usually implies fixing a flaw.	defect, weakness, flaw, fault, bug, issue

and valid text would not be removed. After manual validation, we obtained a total of 50 673 security commit messages from 24 June 1999 to 07 October 2025 written in English, corresponding to the removal of an extra 305 non-English messages and standalone links without any text.

2.2 Informativeness assessment

We now detail the methodology used to extract and classify the different levels of information mentioned in security-related commit messages.

2.2.1 Information extraction. To accomplish this task, we employed named entity recognition (NER), which is a form of natural language processing (NLP) used to extract and identify important information from unstructured text known as entities. The entities can be any word or bag of words that refer to the same entity category. For instance, different forms of games such as "Football", "Cricket", and "Golf" are entities that belong to the same category "Game". However, NER requires the design of specific entity categories and respective entity values, which relies on the target application domain for better accuracy.

In our NER design, we utilized the pretrained `en_core_web_lg` NLP spaCy model, and unlike traditional rule-based approaches, no built-in linguistic rules were applied in the entity recognition process. Instead, we customized the selected model to incorporate a predefined security-specific dictionary collected from the original study and a follow-up work by the same team [25, 26]. The dictionary contains security-specific entity names and categories, which were integrated into the model as a guide for entity identification through pattern matching.

Tables 1 and 2 illustrate the different entity categories, rationales, and their usage in automated patch management triaging systems (i.e., detection, assessment, and prioritization), adapted from the original work. We reused these validated entity categories and definitions to assess information richness in security commit messages. In summary, our NER implementation leverages NLP spaCy model flexibility while relying on the previously verified dictionary, resulting in structured entity recognition from 50 673 security commit messages.

2.2.2 Informativeness classification. To classify security commit message informativeness, we consider six different levels (i.e., Excellent, Very Good, Good, Medium, Poor, and Very Poor), the same used in the original work [25].

Table 3 illustrates the details about the rules based on entity categories extracted by NER, the rationales to classify the informativeness level of security commit messages, and the automated patch management tasks that could be performed.

Information levels of security commit messages are then calculated based on the presence or absence of specific entity categories and rules—again, as it was the case in the original work.

3 Results

In this section, we present the analysis of security commit message informativeness levels to answer our research questions.

3.1 RQ_1 To what extent can we replicate the results of Reis et al. [25] about the informativeness of security-related commit messages?

Motivation: The objective of this question is to confirm the results of the original work for the same code-hosting platform and time period as described.

Approach: To answer RQ_1, we compare the information level of security commit messages between the original results and our replicated-version. Specifically, we examine the consistency of information distribution between the reported results and our replication using security commit messages from the same source (i.e., GitHub) and time period (i.e., from 24 June 1999 to 12 August 2022), as described in the original work. This comparison allows us to assess the validity of the original results, via independent replication (i.e., Replication 1). Finally, we apply the Mann-Whitney U test to statistically compare the results in order to assess the robustness of the conclusion.

Results: For security-related commit messages from the same source (i.e., GitHub) and time span, our replication confirms the results of the original work across all information levels as illustrated in Table 4, with no statistically significant differences observed (p-value 0.4694 > 0.05). This reconfirms that security-related commit messages are, in general, not informative enough for security-focused purposes such as patch triage management.

In particular, the proportions of messages classified as "excellent" (Δ=0.00%), "very good" (Δ=±0.30%), "poor" (Δ=±0.18%), and

Table 2: Tasks in patch triage management systems (from [25]).

Automated Task	Description	Entity category
DETECTION (D)	Detect security-related commits through commit message analysis.	VULNID, ACTION, FLAW, SECWORD
ASSESSMENT (A)	Classify and cluster security-related commits per weakness.	CWEID
PRIORITIZATION (P)	Classify and order security-related commits per severity.	SEVERITY

Table 3: Information spectrum of security commit message (from [25]).

Level	Rule	Rationale	D	A	P
Excellent	$VULNID \wedge CWEID \wedge SEVERITY \wedge SECWORD \wedge ACTION \wedge FLAW$	Include all the different entity categories in a security commit message and enable all the 3 tasks (D, A, P).	✓	✓	✓
Very Good	$VULNID \wedge SECWORD \wedge ACTION \wedge FLAW$	Include all the different entity categories in a security commit message except for metadata (CWEID and SEVERITY). They can still look at the vulnerability report manually to collect the weaknesses type and severity, but not referencing both in the commit message disables A and P.	✓		
Good	$SECWORD \wedge ACTION \wedge FLAW$	only include a description of the vulnerability and respective fix. D is possible, but A and P will fail	✓		
Medium	$ACTION \wedge (FLAW \vee SECWORD)$	Include description of a flaw (that can be a security vulnerability or not) and its respective fix. D is possible but most likely will require manual validation.	✓		
Poor	$VULNID \vee CWEID \vee SEVERITY \vee SECWORD \vee ACTION \vee FLAW$	Include at least one of the entity categories in the security commit message. It may enable one or more of the different tasks (D, A, P), but not all.	✓	✓	✓
Very Poor	No entity was found.	Not included any of the entity categories. D, A, and P tasks fail.			

Table 4: Comparison of results about the informativeness of security-related commit messages across multiple experiments: original work [25], our replication on the same time period and forge (Replication 1), our replication on the same data source but extended time period (Replication 2), our replication on other hosting platforms and extended time period (Replication 3).

Level	**Original result** [Source: GitHub, and Timeline: until 12-08-2022]	**Replication 1** [Source: GitHub, and Timeline: until 12-08-2022]	**Replication 2** [Source: GitHub, and Timeline: until 07-10-2025]	**Replication 3** [Source: Other hosting platforms and Timeline: until 07-10-2025]
Excellent	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Very Good	264 (2.39%)	310 (2.09%)	523 (2.11%)	197 (0.76%)
Good	1262 (11.44%)	1461 (9.86%)	2035 (8.22%)	9267 (35.77%)
Medium	3253 (29.48%)	4811 (32.45%)	7685 (31.03%)	10 804 (41.70%)
Poor	4602 (41.70%)	6155 (41.52%)	10 667 (43.07%)	5158 (19.91%)
Very Poor	1655 (15.00%)	2087 (14.08%)	3854 (15.56%)	483 (1.86%)
Total	11 036 (100%)	14 824 (100%)	24 764 (100%)	25 909 (100%)

very poor (Δ=±0.92%) closely matched with those reported in the original work, indicating a high degree of replicability. However, we observed some minor deviations in the "good" (Δ=±1.58%), and "medium" (Δ=±2.97%) categories, which can be attributed to differences in dataset size, pre-processing, and NER implementation.

Summary of RQ$_1$: *Security-related commit messages are, in general, not informative enough for security-focused purposes when considering the same code-hosting platform (i.e., GitHub) and time period. We confirm, We were able to replicate the original conclusions about the informativeness of security-related GitHub commits from 24 June 1999 to 12 August 2022 with no statistically significant difference.*

3.2 RQ$_2$ Does security commit message informativeness remain consistent over temporal scope and other code-hosting platforms?

Motivation: The objective of this question is to confirm whether the informativeness of security-related commit message remain consistent across extended periods of time (in particular: *after* the period observed in the original study) and across different code-hosting platforms (in particular: elsewhere than GitHub).

Approach: To answer RQ$_2$, we compare the information level of security-related commit messages between the original results and our replication for longer time horizon and other code-hosting

platforms. First, we examine the consistency between the reported results and our replication using security commit messages from the same source (i.e., GitHub) and extended time period (i.e., from 24 June 1999 to 7 October 2025). This comparison allows us to assess whether results remain consistent for the same source and longer time horizon (i.e., Replication 2).

Next, we evaluate results (i.e., Replication 3) on the extended security commit message dataset constructed from other code-hosting platforms (such as GitLab, git.kernel.org, etc.).

Finally, we apply the Mann-Whitney U test to statistically compare these findings in order to assess the robustness for security commit messages sourced from other code-hosting platforms and temporal scope.

Results: For security commit messages from the same source (i.e., GitHub) and extended time span, our replication finds that the informativeness of security-related commit message is *worsening* over time as illustrated in Table 4, with statistically significant differences observed ($p < 0.05$). In particular, we observed a moderate increase in the proportion of "medium" ($\Delta=\pm1.55\%$) and "poor" ($\Delta=\pm1.37\%$) messages, accompanied by a considerable decrease of "good" ($\Delta=\pm3.22\%$) messages.

In addition, results derived from other code hosting platforms (e.g., GitLab, git.kernel.org, etc.) show statistically significant differences ($p < 0.05$) across information levels compared to the original study, while the proportion of "excellent" security commit messages remains the same (i.e., 0.00%). In particular, security commit messages from other code-hosting platforms (such as git.kernel.org) are more frequently classified as "good" ($\Delta=\pm24.33\%$) and "medium" ($\Delta=\pm12.22\%$), with few messages falling under the "very poor" (1.86% only and $\Delta=+13.14\%$) information category compared to messages sourced from GitHub. This finding indicates that security commit messages from other code-hosting platforms are better in terms of content richness compared to GitHub.

Summary of RQ_2: *Over a longer time period, we find that commit-message informativeness is worsening. In addition, we find that security-related commit messages on code-hosting platforms other than GitHub are better in term of informativeness compared to those found on GitHub.*

3.3 RQ_3 Does security commit message informativeness differ significantly by software ecosystems?

Motivation: The objective of this question is to verify whether security-related commit message informativeness differs significantly across software ecosystems.

Approach: To answer RQ_3, first, we map ecosystem names and security commit messages using the commit hashes and software ecosystems specified in OSV vulnerability records (e.g., Android, Ubuntu, Crates, etc.—for a full list of OSV.dev software ecosystems, see the homepage of OSV [4]). We subsequently removed vulnerability records lacking ecosystem names as references and standardized the remaining entries by eliminating any extraneous information (for example, the ecosystem name "Ubuntu:16.04:LTS" referenced in OSV record UBUNTU-CVE-2024-26919 is simplified to "Ubuntu") to ensure consistency. Next, we group security commit messages by ecosystem and compute the proportion of messages at each information level for ecosystems, while excluding ecosystems with fewer than 100 records from the analysis to avoid noise due to statistical measures computed on very small sets.

Finally, we apply the Kruskal-Wallis test to evaluate whether statistically significant differences exist among filtered ecosystems in commit message informativeness. Additionally, we compute mean scores of information levels for each software ecosystem in order to rank them according to the amount of security-related information in commit messages.

Results: Our investigation highlights that security commit message informativeness differs significantly ($p < 0.05$) across the remaining software ecosystems (Linux kernel, Ubuntu, Go, PyPI, etc.) after filtering as summarized in Table 5.

Some ecosystems produce notable portion of good and medium quality messages, while others tend to have more poor and very poor quality descriptions. According to the information level (i.e., mean scores) ranking, operating system-related ecosystems (such as Android, Ubuntu, Linux) demonstrate better security-related commit message informativeness than other ecosystems. In particular, operating system-related ecosystems produce a high proportion of good (32.66% – 39.07%) and medium (40.27% – 47.51%) quality messages, with few messages falling under the very poor (0.53% – 3.53%) information level category. In contrast, other ecosystems (i.e., such as Go, Maven, PyPI) produce more poor (34.53% – 69.6%) and very poor (8.33% – 22.73%) quality messages; however, a few ecosystems (including Go, PyPI, RubyGems, but not all) produce a slightly higher proportion of very good messages (1.02% – 3.09%) than operating system-related ecosystems.

This finding suggests that software projects with long development histories and more formal contribution processes emphasizing clarity and traceability possibly incentivize developers to write more informative security-related commit messages.

Summary of RQ_3: *Commit message informativeness differs significantly across software ecosystems, highlighting ecosystem context as an important factor in determining documentation quality of security-related changes. Overall, security commit messages from operating system-related software ecosystems (e.g., Android, Linux, Ubuntu, etc.) are generally more informative than those from other ecosystems.*

3.4 RQ_4 Does security commit message informativeness differ significantly among commits that comply with the Conventional Commit guidelines and others that do not?

Motivation: The objective of this question is to verify whether commit message informativeness differs significantly between commits that adhere to the emerging guideline for writing commit messages Conventional Commit Specification (CCS) and commits that do not.

Approach: To answer RQ_4, we started by classifying the type of security commit messages into CCS-compliant and non-CCS compliant, using the popularly known Python package conventional_pre_commit [5], in its default configuration (i.e., without customizing the recognized commit types).

Table 5: Comparison of information level assessment of security commit message based on ecosystem.

Ecosystem/ Level	Linux	Android	Ubuntu	Bitnami	Go	Maven	NuGet	Packagist	PyPI	RubyGems	Crates.io	npm
Excellent	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Very Good	27 (0.82%)	9 (0.43%)	194 (0.92%)	5 (2.19%)	30 (2.94%)	17 (0.6%)	6 (2.2%)	16 (1.0%)	56 (3.09%)	8 (2.33%)	1 (0.32%)	10 (1.02%)
Good	1283 (38.91%)	815 (39.07%)	6910 (32.66%)	36 (15.79%)	61 (5.98%)	94 (3.32%)	7 (2.56%)	117 (7.33%)	276 (15.25%)	15 (4.37%)	16 (5.19%)	70 (7.13%)
Medium	1362 (41.31%)	991 (47.51%)	8521 (40.27%)	86 (37.72%)	275 (26.96%)	550 (19.44%)	51 (18.68%)	507 (31.75%)	667 (36.85%)	116 (33.82%)	83 (26.95%)	327 (33.3%)
Poor	599 (18.17%)	260 (12.46%)	4786 (22.62%)	82 (35.96%)	476 (46.67%)	1553 (54.9%)	190 (69.6%)	629 (39.39%)	625 (34.53%)	153 (44.61%)	138 (44.81%)	426 (43.38%)
Very Poor	26 (0.79%)	11 (0.53%)	747 (3.53%)	19 (8.33%)	178 (17.45%)	615 (21.74%)	19 (6.96%)	328 (20.54%)	186 (10.28%)	51 (14.87%)	70 (22.73%)	149 (15.17%)
Total	3297 (100.0%)	2086 (100.0%)	21 158 (100.0%)	228 (100.0%)	1020 (100.0%)	2829 (100.0%)	273 (100.0%)	1597 (100.0%)	1810 (100.0%)	343 (100.0%)	308 (100.0%)	982 (100.0%)

Table 6: Comparison of information level between CCS (Conventional Commit Specification) and non-CCS compliant security commit message.

Level	CCS	Non-CCS
Excellent	0 (0.00%) =	0 (0.00%) =
Very Good	11 (0.59%) ↓	709 (1.45%) ↑
Good	146 (7.89%) ↓	11 156 (22.85%) ↑
Medium	686 (37.08%) ≈	17 803 (36.46%) ≈
Poor	920 (49.73%) ↑	14 905 (30.53%) ↓
Very Poor	87 (4.70%) ↓	4250 (8.70%) ↑
Total	1850 (100%) =	48 823 (100%) =

This resulted in the identification of a total of 1850 (3.65%) CCS-compliant and 48 823 (96.35%) non-CCS-compliant messages. Afterward, we group security commit messages by CCS and non-CCS type and summarized their proportion of classified information level categories.

Finally, we conduct the Mann-Whitney U test to determine whether statistically significant differences exist in commit message informativeness between CCS-compliant and non-CCS-compliant messages.

Results: The results summarized in Table 6 indicate that CCS-compliant security commit messages are *less informative* compared to non-compliant ones, with statistically significant differences observed overall ($p < 0.05$). In particular, CCS-compliant messages are more frequently classified as "medium" (Δ=±0.62%) and "poor" (Δ=±19.20%) quality. In contrast, non-CCS messages exhibit a higher proportion of "very good" (Δ=±0.86%) and "good" (Δ=±14.96%) information levels.

However, a smaller portion of CCS-compliant messages fall into the "very poor" (4.70%) information category compared to non-CCS security commit messages (8.70%), indicating the necessity of explicit rule enforcement to document any security-related changes and raising awareness about the benefits of content richness among developers.

Summary of RQ$_4$: *CCS-compliant security commit messages are less informative compared to non-compliant ones, indicating that CCS guidelines alone appear insufficient to raise information levels although they are intended to standardize and improve the quality and usefulness of commit messages.*

4 Implications and recommendations

Our analysis strengthen previous results reporting that security commit messages are not informative enough for security-focused purposes, such as prototyping an effective patch triage management system.

Further analysis demonstrates that ecosystem context and code-hosting platforms are important factors in the informativeness of security-related commit messages, with statistically-significant variations across different platforms.

Moreover, we observed that emerging guidelines for writing "better" commit messages like Conventional Commits are insufficient to raise information levels, at least for security-related purposes like those analyzed in this study. These findings highlight the necessity of conducting cross-software ecosystem studies to better understand differences in existing practices and to support the development of standard guidelines applicable to all ecosystems for writing security commit messages with information richness.

In summary, it is still necessary to ensure security commit messages are detailed, well-organized, and clearly explain the context of the code changes. Below, we present the key recommendations obtained from the analysis results, aligning with previous studies, for practitioners (i.e., developers, maintainers), researchers, and educators to improve security commit message informativeness and ultimately better serve security-relevant purposes.

4.1 Practitioners

We recommend that practitioners follow strict rules and lexicons when writing commit messages, ensuring semantic alignment with code changes [17, 35]. For instance, a univocal language can be used to identify the type of operation committed by any author, such as security fix, refactor, etc. [17, 35]. Such semantic alignment will help other practitioners understand and extract necessary information for efficient patch management.

To improve automated security patch identification, practitioners should also avoid explicitly hiding the nature of security vulnerabilities in commit messages [22].

Besides, messages should clearly describe the type of security issue addressed by including references (such as CVE or bug IDs) to facilitate easier linking of fixes [30, 33, 37]. In addition, Morrison et al. [21] suggested that combining standard security keywords (e.g., "encryption," "authentication") with project-specific vocabulary further improves precision in identifying security-relevant changes. Furthermore, keeping commit messages concise with separate commit-info, commit-subject, and commit-body can improve automated understanding and accuracy of patch identification [36]. Additionally, senior authors (e.g., maintainers) of a project should be proactive in reviewing commits made by new authors carefully and ensure that code changes are accompanied by well-structured and detailed messages [29].

In summary, clear understanding of these recommendations by practitioners can help them gain the necessary technical knowledge to write better security commit messages, thereby better serving security-related purposes in a rapidly shifting cybersecurity landscape.

4.2 Researchers

Researchers should investigate ways to standardize templates and formal guidelines for writing security commit messages by analyzing existing software ecosystem practices. They should encourage practitioners to adopt best development practices for security by highlighting how informative commit messages with structured and consistent formatting can facilitate effective patch triage management systems [25].

Moreover, researchers should investigate ways to reduce manual verification effort, as previous studies further suggest that even when automated classifiers identify security commits, manual expert verification remains essential for ensuring precision [27]. For instance, our NER implementation requires building a security-specific vocabulary dictionary for better entity extraction and information level classification. Besides, building a better entity dictionary relies on expert domain knowledge. Therefore, researchers should investigate ways to build benchmark security dataset and dictionary for better coverage of security name entities.

4.3 Educators

Our findings can be leveraged by educators as a road map to design their full-stack software development courses for target ecosystems. They should emphasize teaching the purpose and impact of writing high-quality commit messages in distributed version control systems like Git.

In addition, instructors can arrange periodical workshops for practical exercises required to write, review, and revise commit messages for reinforcing industry standard practices. Thus, using peer review and continuous instructor feedback on commit messages can significantly improve students' awareness and writing practices.

Moreover, educators can introduce students to the several existing official conventions for writing commit messages and discuss best practices to be followed during the software development phase.

5 Threats to validity

5.1 Internal validity

Internal validity is threatened by the lack of access to the original reproducibility package and implementation details, which makes exact reproducibility of the methodology infeasible. Therefore, we *replicated* the methodology of the original study, based only on its description in the origin paper. However, we communicated via email with the first author of the original work and sought guidance on the NER implementation and entity dictionary, which helped us reduce ambiguity, but full equivalence with the original implementation is not guaranteed.

We classified CCS and non-CCS security commit messages based on the predefined hooks available in the `conventional_pre_commit` package, which may cause CCS-compliant messages with custom hooks to be missed. We do not see this as serious threat because, if anything, for security purposes we need general guidelines that allow to extract security-relevant information, independently from project-specific customs. Our results provide a first empirical verification of the fact that we are not there yet.

In addition, to extract security commit hashes, we performed manual analysis of sample vulnerability records to formulate better regular expressions, which may also introduce selection bias and consequently influence the final results.

5.2 Construct validity

Construct validity is threatened by dataset reconstruction. Since we did not find data extraction scripts, the constructs had to be redefined using newly curated data and custom extraction techniques.

In addition, to collect the security commit hashes, we considered commit links mentioned in vulnerability record to systematically be references to vulnerability-*fixing* commits, whereas in some cases they may point to vulnerability-*inducing* commits, for documentation purposes. We share this limitation with the original study, which hence does not invalidate the usefulness of the replication part of this study.

Even with guidance from the original authors about the NER entity dictionary, the developed regular expressions may not fully capture the same constructs intended in the original study.

5.3 External validity

External validity is threatened by the observed differences in results when applying the methodology to different data sources. While nearly identical results were obtained on the same data source, inconsistencies across different code-hosting platforms may occur due to data-source dependency.

6 Conclusion

In this paper, we independently replicate a prior original work (Reis et al., 2023) on security commit message informativeness using the only information provided, without reusing any of the original artifacts (data, analysis pipeline, etc.) to assess the robustness of their negative result as reported. In detail, we perform a large scale replication study on 50 673 security commit message dataset (from 24 June 1999 to 07 October 2025) sourced from Software Heritage to assess security commit message informativeness and whether content richness significantly differ by software ecosystem and standard CCS official guideline.

Our replication confirms the original findings in a statistically significant way: security-related commit messages are, in general, not informative enough for security-focused purposes. We also find that commit-message informativeness is worsening over time; while considering GitHub. Further, breaking results down by software ecosystem (Linux kernel, Ubuntu, Go, PyPI, etc.), we observe significant differences in informativeness. In addition, CCS guidelines alone appear insufficient to raise informativeness of security commit message. Based on the findings, we provide recommendations for practitioners, researchers, as well as educators.

This study provides motivation to develop strategies for improving security commit messages. We believe that enriching security

commit messages and maintaining this knowledge base should become a coordinated industry-wide effort, bringing benefits for IT organizations in rapidly shifting cybersecurity landscape.

Data availability

A publicly available reproducibility package [18] containing the curated vulnerability records with patch references, the associated security commit message dataset from the Software Heritage database, a summary of results, and all scripts used in this study are included in the package.

Acknowledgments

This work was supported by France Agence Nationale de la Recherche (ANR), program France 2030, reference ANR-22-PTCC-0001.

References

[1] [n. d.]. ACM Artifact Review and Badging. https://www.acm.org/publications/policies/artifact-review-and-badging-current. Last Accessed: February 25, 2026.
[2] [n. d.]. Conventional Commits. https://www.conventionalcommits.org/en/v1.0.0/. Last Accessed: December 10, 2025.
[3] [n. d.]. NVD (National Vulnerability Database). https://nvd.nist.gov/. Last Accessed: December 23, 2025.
[4] [n. d.]. OSV (Open-Source Vulnerability Database). https://osv.dev/. Last Accessed: December 23, 2025.
[5] [n. d.]. Python package (conventional-pre-commit). https://pypi.org/project/conventional-pre-commit. Last Accessed: February 25, 2026.
[6] [n. d.]. Python package (langdetect). https://pypi.org/project/langdetect/. Last Accessed: February 25, 2026.
[7] Ahmad Abdellatif, Mairieli Wessel, Igor Steinmacher, Marco A Gerosa, and Emad Shihab. 2022. BotHunter: An approach to detect software bots in GitHub. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 6–17.
[8] Jean-Francois Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. 2018. Building the Universal Archive of Source Code. *Commun. ACM* 61, 10 (October 2018), 29–31. doi:10.1145/3183558
[9] C Banerjee and SK Pandey. 2010. Research on software security awareness: problems and prospects. *ACM SIGSOFT Software Engineering Notes* 35, 5 (2010), 1–5.
[10] Russell Brandom. 2017. Former Equifax CEO blames breach on a single person who failed to deploy patch. https://www.theverge.com/2017/10/3/16410806/equifax-ceo-blame-breach-patch-congress-testimony. Last Accessed: Feburary 12, 2026.
[11] Natarajan Chidambaram, Alexandre Decan, and Tom Mens. 2023. A dataset of bot and human activities in GitHub. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 465–469.
[12] Natarajan Chidambaram and Tom Mens. 2025. Observing bots in the wild: A quantitative analysis of a large open source ecosystem. In *2025 IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 1–5.
[13] Nesara Dissanayake, Asangi Jayatilaka, Mansooreh Zahedi, and M Ali Babar. 2022. Software security patch management-A systematic literature review of challenges, approaches, tools and practices. *Information and Software Technology* 144 (2022), 106771.
[14] Kelsey R Fulton, Daniel Votipka, Desiree Abrokwa, Michelle L Mazurek, Michael Hicks, and James Parker. 2022. Understanding the how and the why: Exploring secure development practices through a course competition. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1141–1155.
[15] Mehdi Golzadeh, Alexandre Decan, Damien Legay, and Tom Mens. 2021. A ground-truth dataset and classification model for detecting bots in GitHub issue and PR comments. *Journal of Systems and Software* 175 (2021), 110911.
[16] Dan Goodin. 2017. Failure to patch two-month-old bug led to massive Equifax breach. https://arstechnica.com/information-technology/2017/09/massive-equifax-breach-caused-by-failure-to-patch-two-month-old-bug/. Last Accessed: Feburary 12, 2026.
[17] Yuxiang Guo, Xiaopeng Gao, Zhenyu Zhang, Wing Kwong Chan, and Bo Jiang. 2023. A study on the impact of pre-trained model on Just-In-Time defect prediction. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. IEEE, 105–116.
[18] Syful Islam and Stefano Zacchiroli. 2026. Replication Package (On the Informativeness of Security Commit Messages: A Large-scale Replication Study). https://doi.org/10.5281/zenodo.18757725. Last Accessed: February 26, 2026.
[19] Frank Li, Lisa Rogers, Arunesh Mathur, Nathan Malkin, and Marshini Chetty. 2019. Keepers of the machines: Examining how system administrators manage software updates for multiple machines. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. 273–288.
[20] Moritz Mock, Thomas Forrer, and Barbara Russo. 2024. Where do developers admit their security-related concerns?. In *International Conference on Agile Software Development*. Springer, 189–195.
[21] Patrick Morrison, Tosin Daniel Oyetoyan, and Laurie Williams. 2018. Identifying security issues in software development: are keywords enough?. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 426–427.
[22] Giang Nguyen-Truong, Hong Jin Kang, David Lo, Abhishek Sharma, Andrew E Santosa, Asankhaya Sharma, and Ming Yi Ang. 2022. Hermes: Using commit-issue linking to detect vulnerability-fixing commits. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 51–62.
[23] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. 2020. The Software Heritage Graph Dataset: Large-scale Analysis of Public Software Development History. In *MSR 2020: The 17th International Conference on Mining Software Repositories*. IEEE, 1–5. doi:10.1145/3379597.3387510
[24] Sofia Reis, Rui Abreu, Hakan Erdogmus, and Corina Păsăreanu. 2022. SECOM: Towards a convention for security commit messages. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 764–765.
[25] Sofia Reis, Rui Abreu, and Corina Pasareanu. 2023. Are security commit messages informative? Not enough!. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 196–199.
[26] Sofia Reis, Corina Pasareanu, Rui Abreu, and Hakan Erdogmus. 2023. SECOMlint: A linter for Security Commit Messages. *arXiv preprint arXiv:2301.06959* (2023).
[27] Antonino Sabetta and Michele Bezzi. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 579–582.
[28] Arthur D Sawadogo, Tegawendé F Bissyandé, Naouel Moha, Kevin Allix, Jacques Klein, Li Li, and Yves Le Traon. 2022. SSPCatcher: Learning to catch security patches. *Empirical Software Engineering* 27, 6 (2022), 151.
[29] Sho Suzuki, Hirohisa Aman, Sousuke Amasaki, Tomoyuki Yokogawa, and Minoru Kawahara. 2017. An application of the pagerank algorithm to commit evaluation on git repository. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 380–383.
[30] Xin Tan, Yuan Zhang, Chenyuan Mi, Jiajun Cao, Kun Sun, Yifan Lin, and Min Yang. 2021. Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3282–3299.
[31] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *Proceedings of the 44th International Conference on Software Engineering*. 2389–2401.
[32] Christian Tiefenau, Maximilian Häring, Katharina Krombholz, and Emanuel Von Zezschwitz. 2020. Security, availability, and multiple information sources: Exploring update behavior of system administrators. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 239–258.
[33] Shichao Wang, Yun Zhang, Liagfeng Bao, Xin Xia, and Minghui Wu. 2022. Vcmatch: a ranking-based approach for automatic security patches localization for OSS vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 589–600.
[34] Zhendong Wang, Yi Wang, and David Redmiles. 2022. From specialized mechanics to project butlers: The usage of bots in open source software development. *IEEE Software* 39, 5 (2022), 38–43.
[35] Zhengran Zeng, Yuqun Zhang, Haotian Zhang, and Lingming Zhang. 2021. Deep just-in-time defect prediction: how far are we?. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*. 427–438.
[36] Yaqin Zhou, Jing Kai Siow, Chenyu Wang, Shangqing Liu, and Yang Liu. 2021. Spi: Automated identification of security patches via commits. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021), 1–27.
[37] Fei Zuo, Xin Zhang, Yuqi Song, Junghwan Rhee, and Jicheng Fu. 2023. Commit message can help: security patch detection in open source software via transformer. In *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 345–351.