Are security commit messages informative? Not enough!

Sofia Reis
sofia.o.reis@tecnico.ulisboa.pt
INESC-ID & IST/Técnico, U. of Lisbon
Lisbon, Portugal

Rui Abreu
rui@computer.org
INESC-ID & FEUP, U. Porto
Porto, Portugal

Corina Păsăreanu
pcorina@cmu.edu
Carnegie Mellon University
USA

ABSTRACT

The fast distribution and deployment of security patches are important to protect users against cyberattacks. These fixes can be detected automatically by patch management triage systems. However, previous work has shown that automating the task is not easy, in some cases, because of poor documentation or lack of information in security fixes. For many years, standard practices in the security community have steered engineers to provide cryptic commit messages (i.e., patch software vulnerabilities silently) to avoid potential attacks and reputation damage. However, not providing enough documentation on vulnerability fixes can hinder trust between vendors and users. Current efforts in the security community aim to increase the level of transparency during patch and disclosing times to help build trust in the development community and make patch management processes faster. In this paper, we evaluate how informative security commit messages (i.e., messages attached to security fixes) are and how different levels of information can affect different tasks in automated patch triage systems. We observed that security engineers, in general, do not provide enough detail to enable the three automated triage systems at the same time. In addition, results show that security commit messages need to be more informative—56.7% of the messages analyzed were documented poorly. Best practices to write informative and well-structured security commit messages (such as SECOM) should become a standard practice in the security community.

CCS CONCEPTS

• **Software and its engineering** → Software evolution; • **Security and privacy** → **Software security engineering**.

KEYWORDS

Security, Best Practices, Convention, Commit Messages, Patch Management Process

ACM Reference Format:
Sofia Reis, Rui Abreu, and Corina Păsăreanu. 2023. Are security commit messages informative? Not enough!. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3593434.3593481

EASE '23, June 14–16, 2023, Oulu, Finland
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0044-6/23/06.
https://doi.org/10.1145/3593434.3593481

1 INTRODUCTION

Many efforts in the software security engineering field (e.g., security by design, security policies, application security testing) have been made to avoid software vulnerabilities of reaching production. Yet, the number of vulnerabilities and cyberattacks is still growing. Timely patch management (i.e., the fast distribution and deployment of security fixes to users [5, 13, 15]) is one of the most effective and widely recognized strategies for protecting software systems against cyberattacks [13]. One practical example is the Equifax breach [1]: a failure to patch a 2-month-old critical bug in Apache Struts, which led to a sensitive data breach that impacted 143 million US consumers [2]. Security patches can be detected automatically by patch management triage systems. However, those systems are known to be *inefficient* in identifying and prioritizing security patches [5, 12, 16]. Current processes are largely manual (i.e., time-consuming) [5] and prone to ignore important bug fixes such as the one behind Equifax [1]. One of the reasons why triage systems are *inefficient* is the lack of detailed and quality documentation about security fixes at the commit (or patch) level [6].

Problem: Cryptic messages to document security patches! Previous work has shown that security commit messages (i.e., the commit messages attached to the code changes used to patch software vulnerabilities) can be cryptic [17] and inadequate to perform software vulnerability and patch detection [6]. For many years, standard practices in the security community have steered engineers to provide minimalist commit messages to avoid potential attacks and reputation damages [17]. While this practice is usually used to protect vendors and systems, it also limits the general knowledge pool of people who actually understand the vulnerability and know how to exploit it, which leaves users and defenders unprotected and unaware. According to the CERT Coordinated Vulnerability Disclosure (CVD) guide, this practice should be avoided since knowing the existence of vulnerabilities and their patches is often the key driver to effective patch deployment [3].

Motivation. Figure 1 shows the security commit message used to document the patch for CVE-2022-35928[1]. Although the message only provides a brief description of the patch, it is clear the code changes fix a "security issue" related to "passwords". In reality, this is the patch for a potential buffer overrun vulnerability resulting from reading user-provided passwords and confirmations via command-line prompts—which could be clearer in the message. In addition, the patch fixes an "Improper Authentication" weakness (CWE-287) with a severity (CVSS) score of 8.4 in 10—which is not explicit in the message. This extra information could have helped automated or manual patch triage systems to prioritize this patch since it has high severity. Again, while some argue that security-related documentation should be minimalist, others argue that details are

[1]https://nvd.nist.gov/vuln/detail/CVE-2022-35928

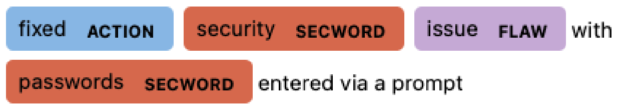

Figure 1: Example of commit message used to patch the software vulnerability identified by CVE-2022-35928

crucial for making automated tools effective [12, 16], creating trust amongst users [3], and enabling fast patch management [3, 16].

Study. Previous work has shown that security-related commits can include relevant information [7, 8, 10]. In this study, we assess how informative security commit messages are and show how the different levels of information affect the different tasks in automated patch triage systems (detection, assessment, and prioritization). We used a Named Entity Recognition (NER) tool to extract security-related information from security commit messages [10]. An example of the extraction result can be seen in Figure 1. We inferred the information level by applying a set of rules on top of the information extracted for each message by the NER tool. In total, we analyzed 11036 security commit messages.

Results. We observed that security engineers provided acceptable or good levels of detail for 44.3% of the security commit messages. However, 56.7% of the messages were poorly documented, which shows security engineers are not writing informative security commit messages yet.

Solution. Detecting and extracting relevant information about software vulnerabilities can be done through vulnerability reports (after *disclosing* time). However, the average time between the availability of a security patch and its disclosure is one week (it varies from days to years) [4], which is a lot of time for users to be exposed to vulnerabilities when a fix is already available. Therefore, it is important to craft informative commit messages to 1) ensure triage systems can automatically detect security patches and trigger notifications to the users about vulnerabilities and respective patches earlier than *disclosing time*, to enable faster patch management and avoid cyberattacks [3, 12, 16]; 2) to improve the human understanding of these software vulnerabilities for researchers and developers, which can drive science forward in the Software Vulnerability Management field [8, 12]. To craft well-structured and informative commit messages, best practices, such as SECOM [9, 10] (a standard for security commit messages validated by the open-source security community) and SECOMlint [11] (a tool to measure compliance against SECOM) should be applied.

Contribution. An empirical analysis of the information included in security commit messages and how different levels of information can affect the different automated tasks performed by patch management processes.

2 HOW INFORMATIVE ARE SECURITY COMMIT MESSAGES?

In this section, we detail the methodology used to classify the different levels of information and report the respective results.

2.1 Dataset Collection and Preprocessing

We leverage the dataset of security commit messages and respective data extraction produced in previous work [10]. This section briefly summarizes the collection and preprocessing steps. The dataset considers data released until the 12th of August, 2022.

2.1.1 Vulnerability Metadata Collection from Public Vulnerability Databases. Public vulnerability databases, such as the National Vulnerability Database (NVD)[2], and the Open-Source Vulnerability (OSV) database[3], integrate documentation (or reports) for thousands of known vulnerabilities. Our dump of the OSV database includes a total of 30091 vulnerability reports for open-source vulnerabilities from different ecosystems: GitHub Advisories, Linux, PyPI, NPM, OSS-Fuzz, Maven, RubyGems, Go, and more. For NVD, we collected a total of 181614 vulnerability reports. In total, we collected 211705 vulnerability reports from both databases.

2.1.2 Collection and Preprocessing of References to Security Patches. Each vulnerability report for both data sources includes a section referencing the fix (or patch) when available.

Collection: To get the commits involved in patching the security vulnerability, we filtered out all the vulnerability reports without references to commit links. We discovered that only 9% of the vulnerability reports in the OSV and NVD have fixes available. In this study, we only focus on vulnerability reports that include commits for fixes available on GitHub—since security patches available through GitHub reflect more than 80% of the commits extracted from vulnerability reports. In total, we found references to GitHub fixes in 8670 NVD reports and 9576 OSV reports.

Preprocessing: To collect the commit message of these commits, we used the GitHub API, which requires knowing the *owner* of the repository which integrates the commit; the *name of the repository*; and the *version* (or, SHA key) that included the vulnerability. However, sometimes due to the lack of precise information, we could not determine the data required to get the commit message (e.g., when the commit link had master instead of a specific SHA key). Thus, we can not ensure that the current version on master is the version where the vulnerability was detected. Therefore, we removed all the vulnerability reports where we found this issue which resulted in a total of 8405 security patches for NVD (3% of data points) and 9466 security patches for OSV (1% of data points).

Merging and Cleaning: Both data sources were merged after normalization into a dataset of 17871 security patches while keeping the vulnerability reports metadata. Many vulnerabilities are reported in both NVD and OSV. Therefore, we found duplicates between both sources using different heuristics: 1) duplicated entries for security patches but with missing values for vulnerability score in one of the sources (18% of data points); 2) OSV reports contain a field called "aliases" which is a list of IDs of the same vulnerability in other databases. Therefore, we removed all the NVD entries (19% of data points) whose IDs were already in the aliases of OSV reports; 3) vulnerabilities fixed with the same patch, usually vulnerabilities that affect different codebases and therefore result in different vulnerability reports were also removed (13% of data points). After removing the different types of duplicates, we end up with a dataset of 10254 security patches.

2.1.3 Collection and Preprocessing of Security Commit Messages. Vulnerabilities can be fixed with one commit (single-commit patch)

[2]National Vulnerability Database: https://nvd.nist.gov/
[3]A distributed vulnerability database for Open Source: https://osv.dev/

or more commits (multi-commit patch)—88.6% (9083) of the patches are single-commit patches while the other 11.4% (1170) of the patches are multi-commit patches. From 10254 security patches, we extracted a total of 11809 security commits. GitHub metadata (including the commit message) was collected using the GitHub API. The commit messages were preprocessed in different ways: **(1)** A total of 334 commits (the equivalent to 160 vulnerability reports) were *no longer available* at the metadata collection time. Therefore, they were removed from the dataset. **(2)** We found *duplicated commit messages* resulting from vulnerability reports with references to the vulnerability fix but deployed in different branches. One example is the GHSA-273r-mgr4-v34f[4], which references a commit per branch where the vulnerability was fixed. In these cases, since the commit messages are the same, we only kept one of the commits. Therefore, an extra 270 commits were removed from the dataset—which left us with 11205 security commit messages. **(3)** As in previous work [14], we removed *non-human written message patterns* except when the original commit message is somehow attached to the commit message under analysis, i.e., includes any text generated by humans. One example is the `GHSA-3m93-m4q6-mc6v`[5] advisory, which only references the cherry-picked commit. In addition to the patterns mentioned in [14], we also removed commit messages with pull request merges from dependabot and pull requests merges without any human text. In summary, we found and removed a total of 126 automated commit messages and kept 11079 security commit messages. **(4)** We noticed some of the commit messages were *not written in English*. Therefore, we ran `langdetect`[6] to infer the message's text language. The model detected 1311 (11.8%) security commit messages as non-English. The tool can perform inaccurate predictions when evaluating too short or too ambiguous text. Therefore, we manually inspected the non-English messages to make sure we would not remove English and valid messages. After manual validation, we removed an extra total of 43 non-English messages, such as "用户头像上传格式限制".

Results. We successfully collected a total of 11036 security commit messages, the equivalent to 9943 security patches.

2.2 Information Level Classification

This section explains the methodology used to classify the different levels of information provided in security commit messages and the respective results.

2.2.1 Named Entity Recognition. NER is a form of Natural Language Processing (NLP) and also referred to as entity chunking, extraction, or identification. It is the task of identifying and extracting key information, called *entities*, from unstructured data (in this case, text). An *entity* can be any word or bag of words that refers to the same *entity category*. For instance, different names of companies, "Netflix", "Google" or "Apple" are entities that belong to the *Company* category. NER requires the design of specific entity categories and the respective entity values, which relies on good domain knowledge.

[4]https://github.com/advisories/GHSA-273r-mgr4-v34f
[5]https://github.com/advisories/GHSA-3m93-m4q6-mc6v
[6]`langdetect`, tool to infer the text language. It supports 55 different text languages.

Table 1: Different Categories of Entities [10].

Category	Rationale	Example
VULNID	Vulnerability IDs are used to identify vulnerabilities for different ecosystems in commit messages: CVE, GHSA, OSV, PyPI, etc. The 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	Vulnerabilities usually belong to a weakness type 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	Vulnerabilities usually have a severity assigned 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

Table 2: Tasks in Patch Management Triage Systems.

Automated Task	Description	Entity Category
DETECTION (**D**)	Locate or 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

2.2.2 Entity Extraction. Previous work has developed a list of category entities (or information) to extract from security commit messages [10]. Table 1 describes the different entity categories we picked from the original list for this study. The rationale behind each category is presented in Table 1. We chose VULNID, CWEID, and SEVERITY because we argue to be the most effective for performing detection (D), assessment (A), and prioritization (P) tasks in commit messages, respectively (Table 2). SECWORD, ACTION, and FLAW categories can also be useful for detecting security-related commits. Yet, not as effective as the VULNID since the vulnerability ID clearly identifies a known security vulnerability. The different entities were extracted from a total of 11036 security commit messages. In this study, we leverage the extraction performed in previous work[10] to assess the different levels of information security engineers provide in current security patch documentation.

2.2.3 Information Level Assessment. We considered six different levels of information for security commit messages: Excellent, Very Good, Good, Medium, Poor, and Very Poor. Table 3 provides details on the rationale behind each level of information, the rule used to classify each level, the number of commit messages classified per level, and which automated tasks could be performed for the different levels of information in security commit messages. Information level classification is calculated based on the presence or absence of specific category entities and combinations.

2.2.4 Results. Our study shows security engineers provide medium or good levels of detail in 44.3% of security commit messages used

Table 3: Information Spectrum of Security Commit Messages.

Level	Rule	Rationale	D	A	P	#Messages
Excellent	VULNID ∧ CWEID ∧ SEVERITY ∧ SECWORD ∧ ACTION ∧ FLAW	Security engineers provide all the different entity categories in a security commit message and **enable all the 3 different tasks** (D, A, P).	✓	✓	✓	0
Very Good	VULNID ∧ SECWORD ∧ ACTION ∧ FLAW	Security engineers provide 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.	✓			264
Good	SECWORD ∧ ACTION ∧ FLAW	Security engineers only provide a description of the vulnerability and respective fix. D is possible, but A and P will fail.	✓			1262
Medium	ACTION ∧ (FLAW ∨ SECWORD)	Security engineers provide a 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.	✓			3253
Poor	VULNID ∨ CWEID ∨ SEVERITY ∨ SECWORD ∨ ACTION ∨ FLAW	Security engineers 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.**	✓	✓	✓	4602
Very Poor	No entity was found.	Security engineers fail to include any of the entity categories. D, A, and P tasks fail.				1655

to patch known security vulnerabilities. However, in none of the cases, patch triage systems could perform all three different automated tasks (D, A, and P). This emphasizes the need for complete security commit messages, i.e., messages that include all six category entities, specially VULNID, CWEID, and SEVERITY, to enable and improve automated tasks in triage systems. A total of 56.7% of the messages were poorly documented. In fact, 15% did not include any of the six category entities, i.e., would be completely missed by automated triage systems. In 41.7% of the poorly documented security commit messages, security engineers provided at least one of the category entities, which can enable some of the different automated tasks but not all as in the excellent information level. The detection (D) task seems to be the easiest to perform (possible in different levels of information, Table 3).

Answer. Not informative enough. Security engineers must provide more detailed documentation to make triage systems effective.

3 TRANSPARENCY IS RISKY BUT NECESSARY

Transparency is a double-edged sword. It can be a source of trust for consumers [3], but also a place for vulnerability [17]. The reality is that non-transparent processes make timely and automated patch management a challenge (or even impossible). The CERT Coordinated Vulnerability Disclosure (CVD) guide suggests avoiding poorly informative documentation since it hinders public awareness of fixes to software vulnerabilities—which breaks user trust in vendors and leads to poor triage systems. The SECOM convention [10] and respective compliance tool [11] are meant to help vendors to produce informative documentation for security patches at the commit level. We are aware that being too much transparent can make users vulnerable to cyberattacks, but we argue that providing better documentation will help triage systems locate those fixes instantly and enable fast security patch management processes. Thereby, users and companies can benefit from a solution that notifies them as soon as a new security patch is released, which is usually one week earlier than the disclosure [4]. **Risk Analysis.** Experienced security engineers should determine the level of detail to provide. If development systems are private, then no major risks are attached to providing detailed security commit messages (in principle, we do not account for internal attackers). However, when a critical vulnerability is to be patched in open-source software of wide use, then security engineers should carefully decide which details to include in the message. They should provide enough information to users to understand its criticality but not enough to attackers.

4 CONCLUSIONS

In this study, we assess the different information levels in security commit messages to understand how detailed and transparent patch documentation at the commit level is and how the level of information can affect automated triage systems (important to avoid cyberattacks). We observed that security commit messages must be more informative to enable detection, assessment, and prioritization (all important tasks for security maintenance teams).

REFERENCES

[1] 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.

[2] 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/.

[3] Allen Householder, Garret Wassermann, Arthur Manion, and Christopher King. 2020. CERT® Guide to Coordinated Vulnerability Disclosure. (9 2020). https://doi.org/10.1184/R1/12367340.v1

[4] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *CCS'17*. 2201–2215.

[5] 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 *SOUPS @ USENIX'19*.

[6] Serena Elisa Ponta, Henrik Plate, and Antonino Sabetta. 2018. Beyond Metadata: Code-centric and Usage-based Analysis of Known Vulnerabilities in Open-source Software. (2018).

[7] Serena E. Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 2019. A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software. In *MSR'19*. IEEE Press, 383–387.

[8] Sofia Reis and Rui Abreu. 2017. SECBENCH: A Database of Real Security Vulnerabilities.. In *SecSE @ ESORICS*. 69–85.

[9] Sofia Reis, Rui Abreu, Hakan Erdogmus, and Corina Păsăreanu. 2022. SECOM: Towards a convention for security commit messages. In *MSR'22*.

[10] Sofia Reis, Hakan Erdogmus, Rui Abreu, and Corina Pasarenau. 2023. Best Practices when Writing Security Commit Messages: Are we there yet?

[11] Sofia Reis, Corina Pasareanu, Rui Abreu, and Hakan Erdogmus. 2023. SECOMlint: A linter for Security Commit Messages.

[12] Arthur D. Sawadogo, TegawendéF. Bissyandé, Naouel Moha, Kevin Allix, Jacques Klein, Li Li, and Yves Le Traon. 2022. SSPCatcher: Learning to catch security patches. *Empirical Software Engineering* 6 (2022), 151.

[13] Murugiah Souppaya and Karen Scarfone. 2013. Guide to Enterprise Patch Management Technologies. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r3.pdf.

[14] Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. 2022. What makes a good commit message?. In *ICSE'22*. ACM.

[15] Christian Tiefenau, Maximilian Häring, Katharina Krombholz, and Emanuel Von Zezschwitz. 2020. Security, Availability, and Multiple Information Sources: Exploring Update Behavior of System Administrators *(SOUPS'20)*.

[16] Zheng Zhang. 2021. An Investigation of the Android Kernel Patch Ecosystem. In *USENIX'21'*.

[17] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *ASE'21*.