

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Enhancing Security Patch Documentation Through Generative AI

Iarina Majeri



Masters of Software Engineering

Supervisor: Rui Filipe Lima Maranhão de Abreu

Co-Supervisor: Sofia Oliveira Reis

June 30, 2025

© Iarina Majeri, 2025. This thesis was written by the author in partial fulfillment of the requirements for the Master's degree. It benefited from the use of AI-assisted tools, including OpenAI's ChatGPT, for non-substantive support such as improving English clarity. All scientific content, experimental results, and interpretations are the original work of the author.

Enhancing Security Patch Documentation Through Generative AI

Iarina Majeri

Masters of Software Engineering

June 30, 2025

Abstract

Security patch documentation is a critical yet time-consuming aspect of secure software development. This thesis investigates the use of generative AI models to automate the generation of SECOM-compliant commit messages directly from code diffs. Two prompting strategies are evaluated: a zero-shot baseline, where messages are generated from raw Git diffs without examples, and a few-shot approach, where structured exemplars guide the model via in-context learning.

In a large-scale benchmark of 500 real-world security patches, zero-shot prompting achieved a SECOMLint structural compliance rate of 33%, surpassing the 6.8% compliance rate of original human-written messages, a relative improvement of +383%. However, metadata accuracy remained low, highlighting the limitations of unguided generation. A targeted evaluation of few-shot prompting across 215 entries drawn from five CWE categories revealed substantial relative gains: format compliance increased by 11.84%, metadata accuracy by 34.08%, and CVSS exact match rate by 177.57%, while severity estimation error (MAE) decreased by 15.72%.

The results demonstrate that few-shot prompting significantly enhances both structural and semantic fidelity, especially for well-represented vulnerability classes. However, performance remains sensitive to data scarcity and semantic complexity, as shown in the underperformance for CWE-918. In addition to contributing a replicable prompting framework and validation pipeline, this work introduces a vision for SECOMLint as a model-assisted documentation tool. The findings highlight the promise and boundaries of using large language models to support structured security documentation and lay the foundation for future tooling that reduces the manual burden of writing security-aware commit messages.

Keywords

Security Commit Messages; SECOM Standard; Generative AI; Learning Approaches.

Acknowledgments

I would like to begin by thanking Professor Rui Filipe Lima Maranhão de Abreu for his thoughtful guidance and expertise throughout the development of this thesis. His consistent support and valuable insights have been crucial in shaping both the direction and quality of this work.

I am also truly grateful to Sofia Reis for her generosity with her time, her encouragement, and the way she continuously shared stories that inspired and uplifted me. Her remarkable knowledge and willingness to help were a constant source of motivation.

To Marija Jakovleva — thank you for being the anchor I did not know I would need. From the day we met during the master's programme, your friendship has felt like home.

Finally, my deepest thanks go to my family and fiancée. Even from afar, they never stopped believing in me. Every milestone I have reached here has only been possible because of their unshakable support.

“We are drowning in information but starved for knowledge. ”

John Naisbitt

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Problem Statement	3
1.3	Contributions	4
1.4	Dissertation Structure	4
2	Literature Review	6
2.1	The Role of Documentation in Software Engineering	6
2.2	Generative AI in Documentation	7
2.3	Challenges in Security Patch Documentation	8
2.4	Related Work	9
2.4.1	SECOM and SECOMlint	9
2.4.2	Tools for Automatic Documentation Generation	12
3	Empirical Study Design and Dataset	15
3.1	Research Questions	15
3.2	Dataset Overview	16
3.2.1	Data Collection and Sources	16
3.2.2	Filtering Heuristics	17
3.2.3	Subset Construction and Management	17
3.2.4	Data Schema and Fields	18
3.2.5	Dataset Statistics and Characteristics	20
4	Zero-Shot Generation of SECOM-Compliant Security Commit Messages	24
4.1	Model and Prompting Configuration	24
4.2	Data Processing and Message Generation Pipeline	26
4.3	Validation Strategy and Results Analysis	29
4.3.1	Validation Strategy	29
4.3.2	Results: SECOMLint Compliance	33
4.3.3	Results: Metadata Accuracy	35
5	Improving SECOM Metadata Accuracy with Few-Shot Learning	40
5.1	Motivation and Metadata Focus for RQ2	40
5.2	Data Processing and In-Context Learning Pipeline	43
5.2.1	Prompting Strategy	45
5.3	Validation Strategy and Results Analysis	47
5.3.1	Validation Strategy	47
5.3.2	In-Context Learning Approaches Results	48

6	Conclusions and Future Work	54
6.1	Conclusions	54
6.2	Future Work	55
	References	57
A	Few-shot User Prompt	62
B	System Prompt	67
C	Public Repository - Thesis Code	69

List of Figures

2.1	Week-wise average commit message quality	8
2.2	Structure of a SECOM commit message	10
2.3	Example of a commit message used in patching CVE-2022-35928 (screenshot taken on August 10th 2024)	11
2.4	Execution steps of SECOMlint and its different components	12
3.1	Pie chart showing distribution of single vs multi-commit patches	20
3.2	Bar chart showing the distribution of patch sizes.	21
3.3	Pie chart showing distribution of vulnerabilities by CVSS severity level	21
3.4	Top 15 most frequent CWE categories in the dataset	23
3.5	Number of security patches over time (2010—2022)	23
4.1	CVSS score match between generated message and ground truth.	35
4.2	Severity level match between generated message and ground truth.	36
4.3	CWE weakness match between generated message and ground truth.	36

List of Tables

1.1	Examples of real security commit patches	2
2.1	Entities description and rationale.	11
3.1	Top 15 CWE Categories in the Dataset	22
4.1	SECOMLint rules and corresponding validation criteria	30
4.2	Average compliance scores for original and generated messages	33
4.3	Example of original vs. generated commit message and SECOMLint scores	34
4.4	Average BLEU and METEOR scores for generated and original messages	37
5.1	SECOM format components grouped by category, with generability and retrieval comments.	41
5.2	Comparison of zero-shot and few-shot prompting on the merged evaluation dataset	48
5.3	Evaluation results for CWE-617 (Reachable Assertion), 14 entries	49
5.4	Evaluation results for CWE-79 (Cross-site Scripting), 141 entries	50
5.5	Evaluation results for CWE-125 (Out-of-Bounds Read), 34 entries	50
5.6	Evaluation results for CWE-89 (SQL Injection), 17 entries	51
5.7	Evaluation results for CWE-918 (Server-Side Request Forgery), 9 entries	51

Abbreviations and Symbols

AI	Artificial Intelligence
API	Application Programming Interface
BLEU	Bilingual Evaluation Understudy (score)
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
GPT	Generative Pre-trained Transformer
ICL	In-Context Learning
LLM	Large Language Model
MAE	Mean Absolute Error
METEOR	Metric for Evaluation of Translation with Explicit ORdering
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
NMT	Neural Machine Translation
NVD	National Vulnerability Database
OSV	Open Source Vulnerability (Database)
RQ	Research Question
SECOM	Security Commit Message (Convention)
SECOMLint	SECOM Linter (validation tool for commit messages)
SVM	Software Vulnerability Management
URL	Uniform Resource Locator

Chapter 1

Introduction

This chapter introduces the motivation and context for the study, focusing on the challenges of documenting security patches in a way that supports effective vulnerability management. It highlights the limitations of current practices, including the prevalence of uninformative or misleading commit messages, and outlines the need for standardized, structured documentation. The chapter also presents the objectives of the dissertation and frames the use of generative AI as a promising approach to improve the quality and consistency of security commit messages.

The cause of significant cybersecurity attacks has been the delay in the deployment of remedies to known software security vulnerabilities (Thomas, 2020; Brandom, 2017; News, 2022; Cybersecurity and Agency, 2022; Security, 2019). Timely patch management, which involves the rapid distribution and dissemination of security fixes to users (Souppaya and Scarfone, 2013; Li et al., 2019; C.9 et al., 2020; ref, 2019), is one of the most effective and widely recognized strategies for safeguarding software systems from cyberattacks (Security, 2019; ref, 2019). One significant challenge remains: the absence of efficient patch triage systems for identifying and prioritizing security patches. Current processes are predominantly manual, which is time-consuming and prone to overlooking critical bug fixes. A large scale empirical analysis of over 4,000 security patches across 682 open-source projects found that many vulnerabilities persist for weeks to months post-disclosure, making systems vulnerable to attack even when patches exist, indicating that manual triage and deployment remain slow and ineffective. (Li and Paxson, 2017).

A recent study indicated that the informativeness of commit messages associated with security patches may impede the effectiveness of triage system tasks, such as detection, prioritization, and assessment (Reis et al., 2023a). Prior research has examined the utilization of patch metadata, such as commit messages, alongside code modifications to investigate automated patch detection (Sawadogo et al., 2022; Reis and Abreu, 2017; Zhou et al., 2021a; Ponta et al., 2018). The findings indicate that software vulnerability management (SVM) techniques cannot depend solely on metadata for the identification of software vulnerabilities, as the data is frequently inaccurate and incomplete (Ponta et al., 2018). Only 38% of commit messages utilized to “silently” patch software vulnerabilities in the past contained security-related terminology (Zhou et al., 2021a). Silent fixes occur when a vendor addresses a vulnerability without disclosing its existence in any

documentation, such as release notes or commit messages (Zhou et al., 2021a).

This practice results in reduced informative patch documentation and impairs the awareness and effectiveness of triage systems. This practice typically safeguards vendors and systems; however, it restricts the broader understanding of vulnerabilities and exploitation methods, ultimately leaving users and defenders vulnerable and uninformed. Silent patches should be avoided, as stated in the CERT Coordinated Vulnerability Disclosure (CVD) guide (Tian et al., 2022). This is due to the fact that knowing the existence of vulnerabilities and the patches for those vulnerabilities is frequently the most important factor in determining how effectively patches are deployed.

Table 1.1: Examples of real security commit patches

	Commit Message	VulnID	SHA
1	.	CVE-2019-13568	ac80033
2	Minor patch.	OSV-2020-2108	a8bf10e
4	<scratchsig> <script>location= 'https://www.youtube.com/watch?v=dQw4w9WgXcQ';</script> <scratchsig>	CVE-2020-15179	4160a39
5	Fixed security issues with passwords entered via a prompt	CVE-2022-35928	6876185

This thesis proposes SECOM (Security-Enhanced Commit Message), a structured format designed to enrich security patch commit messages with standardized metadata. A detailed specification is provided in Section 2.4 SECOMLint is a linting tool designed to validate SECOM-compliant commit messages based on a set of syntactic and semantic rules, detailed in the same section.

1.1 Motivation

“It is not enough for code to work. It must also be readable and understandable to others, because software is a human artifact, and humans must maintain and evolve it.” (Spinellis, 2003, pg. 18)

Security commit messages, which are associated with code changes that address software vulnerabilities, often exhibit a cryptic nature (Zhou et al., 2021a). Table 1.1 presents examples of security commit messages (column *Message*) utilized to document patches addressing known software vulnerabilities (column *VulnID*) alongside the corresponding patch commit key (column *SHA*). Message 1 (“.”) exemplifies a cryptic message, as it lacks informative content. Messages indicate that a code defect is being addressed, but no further information is available. Message 3 refers to an internet meme known as “Rickrolling,” which is typically employed to prank individuals by directing them to the song “Never Gonna Give You Up.” The message, while humorous,

fails to convey pertinent information about the vulnerability and its corresponding patch. The majority of cases discussed are likely to bypass detection by triage systems, resulting in users remaining uninformed about new patches and vulnerable to security threats. Message 4 offers a succinct overview of the patch and pertinent details, including its resolution of a security issue associated with passwords. The patch addresses an “Improper Authentication” vulnerability (CWE-287) with a severity score of 8.4 out of 10, which is not clearly stated in the message.¹

This example illustrates a broader and persistent issue in security patch documentation: even when a vulnerability is addressed through a corrective patch, the corresponding commit message frequently omits essential metadata and descriptive context. Such omissions hinder the effectiveness of automated vulnerability triage systems and reduce the transparency and traceability of remediation efforts.

1.2 Problem Statement

Thesis Objective: The primary objective of this thesis is to evaluate and improve the use of generative language models for producing structured documentation of security patches. Specifically, it explores how these models can generate SECOM-compliant commit messages and infer critical metadata that supports patch triage, traceability, and secure software maintenance.

Hypothesis: This thesis hypothesizes that generative language models can enhance software security workflows by producing commit messages that conform to SECOM — a structured commit message convention designed to improve traceability, clarity, and vulnerability analysis. By conditioning the model on relevant in-context examples, it is expected to infer and express key security-related metadata with greater accuracy, addressing the challenges outlined in this section regarding the lack of consistency and structure in current commit practices.

This hypothesis is investigated through the following research questions:

- **RQ1:** To what extent can generative AI models produce SECOM-compliant commit messages directly from code diffs, in a zero-shot setting?
- **RQ2:** Can in-context learning strategies enhance the accuracy of structured metadata fields—such as CWE identifiers, severity labels, and CVSS scores—in the generated messages?

These research questions are of practical significance, as they explore the applicability of large language models (LLMs) to a domain where precision, structure, and consistency are essential. The findings may inform future tool development efforts aimed at reducing the manual burden of documentation while improving the quality and standardization of security communication across software ecosystems.

¹“Rickrolling” meme details at <https://knowyourmeme.com/memes/rickroll>

1.3 Contributions

The contributions of this thesis are summarized as follows:

- The construction of a high-quality dataset comprising 500 real-world security patches, annotated with SECOM-compliant metadata and filtered using heuristics tailored to the generative task.
- The design and implementation of a zero-shot prompting strategy for security commit message generation, evaluated using structural compliance metrics and metadata alignment.
- A targeted evaluation of few-shot prompting technique, demonstrating measurable improvements in format adherence and metadata fidelity across multiple vulnerability categories.
- The integration of SECOMLint as a rule-based validation tool to assess the structural and semantic quality of generated messages.
- A reproducible pipeline for commit message generation and validation, made publicly available to support future research and practical adoption.

1.4 Dissertation Structure

This dissertation is organized into six chapters, each contributing to the overarching goal of evaluating and enhancing the generation of structured security patch documentation using generative AI:

- **Chapter 1 — Introduction:** Presents the motivation behind the research, highlighting the challenges of inadequate security commit messages and their implications for vulnerability detection and patch triage. It frames the problem contextually within the broader goals of secure software development.
- **Chapter 2 — Literature Review:** Surveys prior work on commit message generation, documentation standards like SECOM, and related techniques in machine learning and natural language generation. It also introduces SECOMLint as a validation tool.
- **Chapter 3 — Empirical Study Design and Dataset:** Details the research questions guiding the study and the methodology used to construct a high-quality dataset of security-relevant commits. This includes data sourcing, filtering heuristics, schema design, and statistical characteristics.
- **Chapter 4 — Zero-Shot Generation of SECOM-Compliant Security Commit Messages:** Addresses **Research Question 1 (RQ1)**, which investigates whether large language models (LLMs) can generate SECOM-compliant commit messages using a zero-shot prompting strategy. The chapter presents the experimental setup, system and user prompts, validation approach using SECOMLint, and quantitative results on structural and metadata accuracy.

- **Chapter 5 — Improving SECOM Metadata Accuracy with Few-Shot Learning:** Addresses **Research Question 2 (RQ2)**, which explores whether few-shot prompting can improve the accuracy of structured metadata fields in SECOM messages. It introduces a targeted evaluation using commits across five CWE (Common Weakness Enumeration) categories and demonstrates measurable gains in metadata fidelity, severity estimation, and SECOM format compliance compared to the zero-shot baseline.
- **Chapter 6 — Conclusions and Future Work:** Summarizes the key findings from both experimental phases, discusses their implications, and outlines directions for future research, including fine-tuning strategies and enhancements to the SECOMLint tool.

Chapter 2

Literature Review

This chapter reviews foundational and contemporary research relevant to the generation of security commit messages and the use of generative AI in documentation tasks. It begins by contextualizing the role of documentation in software engineering, particularly within collaborative development workflows. The chapter then explores the evolution of commit message generation techniques—from rule-based methods to modern neural approaches—and highlights specific challenges in documenting security patches. It concludes with a detailed overview of the SECOM convention and SECOMLint tool, as well as related work on automatic documentation generation, establishing the background for the empirical studies that follow.

2.1 The Role of Documentation in Software Engineering

Collaborative software development is inherently a social activity, commonly facilitated by version control systems (VCS) such as Git. These systems not only maintain detailed documentation of code changes but also enable simultaneous access to project artifacts. Each commit alters source code and related files, and it is expected to be accompanied by a message that explains both the changes made and the reasoning behind them (Zagalsky et al., 2015; Zhu et al., 2016; Buse and Weimer, 2010b; Mockus and Votta, 2000a; Zhang et al., 2024).

Qualitative analyses suggest that commit messages are expected to address two core elements: what was changed and why. A survey of developers showed that 93% agree commit messages should include both “What” and “Why” content to help collaborators understand the context and intent behind code modifications.

Despite their importance, writing commit messages—especially concise subject lines—is often seen as a difficult and time-consuming task. Surveys show that 52% of developers support the use of automated methods for generating commit message summaries. Furthermore, several studies have found that commit messages frequently lack essential information or are omitted entirely. One investigation revealed that 44% of commit messages fail to meet content expectations, and another analysis of 23,000 projects found that 14% of commit messages were completely empty (Tian et al., 2022; Dyer et al., 2013; Zhang et al., 2024).

This lack of documentation quality poses significant challenges. Commit messages are crucial for communicating the intent and rationale of code changes. In their absence, this knowledge often becomes tacit and risks being lost when developers leave a project (Zhang et al., 2024).

2.2 Generative AI in Documentation

Efforts to improve commit documentation have led to substantial progress in automating commit message generation. Early automation strategies were primarily rule-based or template-driven. For instance, DeltaDoc leveraged symbolic execution to express code modifications, while tools such as ChangeScribe and AutoSumCC focused on summarizing method additions and changes using predefined templates. These approaches were largely effective in capturing the “What” aspect of changes but fell short in conveying the “Why”—a critical element for high-quality documentation (Linares-Vásquez et al., 2015; Cortés-Coy et al., 2014; Liu et al., 2018; Wang et al., 2021; Zhang et al., 2024).

Subsequent techniques adopted information retrieval-based models. NNGen (Liu et al., 2018), for example, uses the Nearest Neighbor algorithm to generate commit messages by matching current code changes with similar historical modifications, evaluating similarity using metrics like cosine and BLEU scores. LogGen (Hoang et al., 2020), another retrieval-based method, employs learned vector representations (such as CC2Vec) to encode code changes. However, these techniques often struggle when no closely related examples exist in the training corpus (Zhang et al., 2024).

More recently, a third class of approaches has emerged based on learning-based techniques that frame commit message generation as a translation task. These neural machine translation (NMT) models take code differences as input and produce natural language descriptions (Jiang et al., 2017; Dong et al., 2022; Liu et al., 2019b; Xu et al., 2019; Loyola et al., 2017, 2018; Liu et al., 2019a; Nie et al., 2021). Notably, FIRA (Dong et al., 2022) models code edits as fine-grained graphs and has demonstrated significant improvements over prior methods, establishing itself as a state-of-the-art solution in commit message automation (Zhang et al., 2024).

Nonetheless, the quality of commit messages remains a persistent concern. A longitudinal study utilizing Spearman correlation analysis (Myers and Sirois, 2004) identified a notable decline in the overall quality of commit messages over time, as measured by “What,” “Why,” and “Good” quality scores. The most pronounced drops were observed in the “What” score (−0.79953) and the “Good” score (−0.80203), while the “Why” score showed a more moderate decline (−0.42674). These findings underscore a systemic reduction in documentation quality across repositories. Weekly quality score distributions further confirm this trend, reinforcing the need for automated solutions to uphold consistency and clarity in commit documentation (Li and Ahmed, 2023). Figure 2.1 illustrates the observed trends.

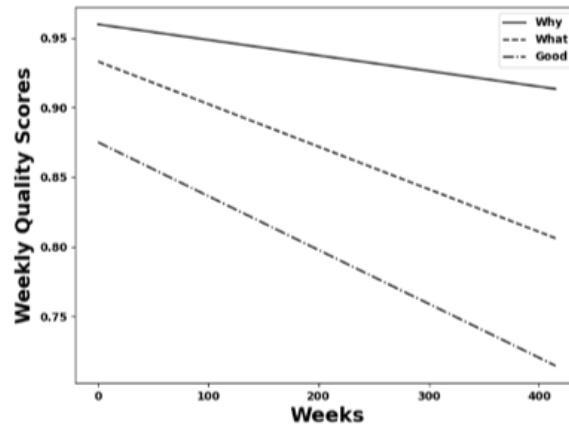


Figure 2.1: Week-wise average commit message quality

2.3 Challenges in Security Patch Documentation

Commit messages associated with security patches, particularly those addressing software vulnerabilities, present unique documentation challenges. Prior research suggests that these messages often lack clarity and sufficient detail, limiting their utility for critical tasks such as vulnerability identification and patch triage (Reis et al., 2023c).

A prevailing practice in the security domain favors brevity in commit messages, aiming to reduce the risk of exposing sensitive information or drawing undue attention to vulnerabilities (Zhou et al., 2021b). While this approach protects against malicious exploitation and reputational harm, it simultaneously restricts access to crucial context for defenders and maintainers. As a result, system administrators and users may remain unaware of significant vulnerabilities, increasing their exposure to potential threats.

The CERT Coordinated Vulnerability Disclosure (CVD) guide underscores the importance of transparency in disseminating information about vulnerabilities and their corresponding patches (Householder et al., 2020). It highlights that timely awareness is often pivotal for effective deployment of security updates. Nonetheless, striking a balance between transparency and operational security remains a persistent challenge (Reis et al., 2023c).

Software security patches, designed to remediate known vulnerabilities, are generally prioritized over other types of updates by both industry professionals and academic researchers. Their ability to prevent exploitation by malicious actors underscores their role as the most reliable strategy for addressing software security issues. Effective patch management is therefore fundamental to preserving the confidentiality, integrity, and availability of IT systems (Mell et al., 2005; C.9 et al., 2020; Araujo et al., 2014; Reis et al., 2025a).

Empirical analysis of commit message quality has further illuminated these concerns. Longitudinal studies employing Spearman correlation (Myers and Sirois, 2004) have demonstrated a decline in message quality over time, particularly in dimensions labeled as “What” ($\rho = -0.79953$) and “Good” ($\rho = -0.80203$), with the “Why” dimension declining more gradually ($\rho = -0.42674$).

These findings reinforce the urgency of developing automated tools that can uphold high documentation standards in security-critical contexts. Recent advancements, such as FIRA, which leverage fine-grained edit graphs and sub-token encoding, have shown promise. However, existing methods continue to fall short in addressing the nuanced demands of security-oriented documentation (Dong et al., 2022).

2.4 Related Work

This section surveys prior work related to secure commit message conventions, structured vulnerability documentation, and the use of language models for code-related generation tasks. The discussion begins with an overview of the SECOM convention and its associated validation framework, SECOMLint, which form the foundation for the evaluation conducted in this thesis.

2.4.1 SECOM and SECOMlint

Sofia Reis et al. noted in (Reis et al., 2025b) that the SECOM convention was developed from established sources (ref; Patterson; Torvalds; Beams) regarding effective commit message writing to promote its adoption. The convention’s structure and fields were derived from two sources: 1) empirical analysis of security-related commit messages, and 2) feedback obtained from presentations to two Open Source Security Foundation (OpenSSF) working groups, namely “Best Practices” and “Vulnerability Disclosure.” The convention comprises several distinct sections. The header contains the type of vulnerability fix, a brief description of the vulnerability, and its identifier when applicable. The body outlines the nature of the vulnerability (what), its implications (why), and the corresponding patch (how). Metadata includes the type of weakness (CWE-ID), severity, CVSS score, detection methods, a link to the report, and the software version in which the vulnerability was introduced. Additionally, it lists authors and reviewers, along with references to bug trackers. Figure 2.2 illustrates the entire structure.

Vulnerabilities are documented on platforms including the National Vulnerability Database (NVD) and the Open-Source Vulnerability (OSV) Database. An example of a recognized vulnerability is CVE-2022-35928. A security commit refers to any commit that addresses a known security vulnerability, exemplified by CVE-2022-35928. The section titled “References to Advisories, Solutions, and Tools” in the CVE-2022-35928 report includes a GitHub reference to commit 6876185, which is responsible for the patch. A security commit message is generally employed to record the security commit implemented to address the vulnerability. Figure 2.3 illustrates the commit message utilized to document the patch for CVE-2022-35928. This commit message offers a succinct overview of the patch. The description lacks detail regarding the vulnerability, its significance, and the corresponding patch. The document lacks essential details such as the CVE identifier, type of weakness, severity of the vulnerability, and information regarding the reviewers and authors of the patch, which are critical for comprehending and managing both the vulnerability and the patch (Reis et al., 2023b).

```

vuln-fix: <header/subject> (<Vuln-ID>)

<body>
# What:   Describe the vulnerability.
# Why:    Explain its impact or risk.
# How:    Describe how it was fixed.
</body>

[For each identified weakness:]
Weakness:  <Weakness Name or CWE-ID>
Severity:  <Low | Medium | High | Critical>
CVSS:      <CVSS Score (0-10)>
Detection: <Detection Method>
Report:    <Link to Advisory or Report>
Introduced-in: <Commit Hash of Introduced Vuln>
[End Weakness Block]

Reported-by:  <Name> (<Contact>)
Reviewed-by:  <Name> (<Contact>)
Co-authored-by: <Name> (<Contact>)
Signed-off-by: <Name> (<Contact>)

Bug-tracker: <Bug Tracker URL>
Resolves:    <Issue or PR Number>
See also:    <Related Issue or PR Number>

```

Figure 2.2: Structure of a SECOM commit message

The convention, as illustrated in Figure 2.2, comprises five primary sections: The header contains the type of vulnerability fix, a brief description of the vulnerability, and its identifier when available. The body outlines the vulnerability (what), its impact (why), and the patch to address the vulnerability (how). Metadata includes the type of weakness (CWE-ID), severity, CVSS, detection methods, report link, and the software version in which the vulnerability was introduced. Contacts list the names and email addresses of the reporters and reviewers. Finally, references to bug trackers are provided. Different sections should be delineated by a new line (Reis et al., 2023b).

Figure 2.4 illustrates the various components of SECOMlint and the information flow during a single execution. The tool accepts a commit message, which is subsequently parsed and segmented into distinct sections: header, body, metadata, contact, and bug tracker references. It utilizes our Named Entity Recognition (NER) extractor for each of these sections. Named Entity Recognition (NER), also referred to as entity chunking, is a method employed in natural language processing to identify and extract significant information, termed entities, from text. Entities can include organizations, individual names, company names, and additional categories. spaCy⁴ offers language processing pipelines that execute various pre-processing steps, including tokenization (segmenting text into tokens), tagging (assigning part-of-speech tags), parsing (assigning dependency labels between tokens), and named entity recognition (detecting and labeling named entities). The various steps may be enabled, disabled, or substituted with new rule-based models; for instance, named entity recognition (NER) can be replaced by novel rule-based models to extract different types of entities.

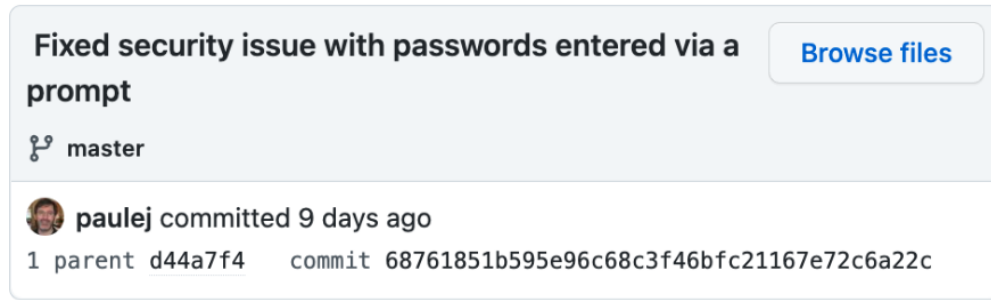


Figure 2.3: Example of a commit message used in patching CVE-2022-35928 (screenshot taken on August 10th 2024)

Table 2.1: Entities description and rationale.

Entity	Rationale
ACTION	A commit usually implies an action: adding some features, fixing a problem, refactoring code, and more.
FLAW	Fixing a security vulnerability usually implies fixing a flaw (e.g., problem, defect, issue, weakness, flaw, fault, bug, error, etc.).
VUNLID	Known vulnerability IDs: CVE, GHSA, OSV, etc.
CWEID	Vulnerability type (CWE-ID/Weakness name).
ISSUE	The GitHub issue number or pull request sometimes is referenced in commit messages.
EMAIL	Contact e-mails of reviewers and authors usually appear after tags such as ‘Reported-by’.
URL	Vulnerability reports or blog posts; and bug-trackers references.
SHA	Sometimes commit hashes are mentioned to reference where the vulnerability was introduced.
VERSION	Software versions are sometimes important to find mentions of malicious software.
SEVERITY	Vulnerability severity: low, medium, high, critical.
DETECTION	Vulnerabilities are detected manually or using specific tools (such as codeql, coverity, oss-fuzz, libfuzzer, and more).
SECWORD	Words or group of words that were identified as security-relevant in previous work Le et al. (2021) ; Croft et al. (2021) .

A comprehensive empirical analysis of security commit messages revealed distinct groups of keywords that may signify various entities, including vulnerability IDs, weakness IDs, severity levels, and other security-related terms. The identified groups of keywords or patterns can be formulated into rules for application in the NER component of the spaCy pipeline to extract the specified entities. Table 2.1 presents the various types of entities that are ultimately designed. Certain entities are not specific to security; however, they are significant as they represent patterns

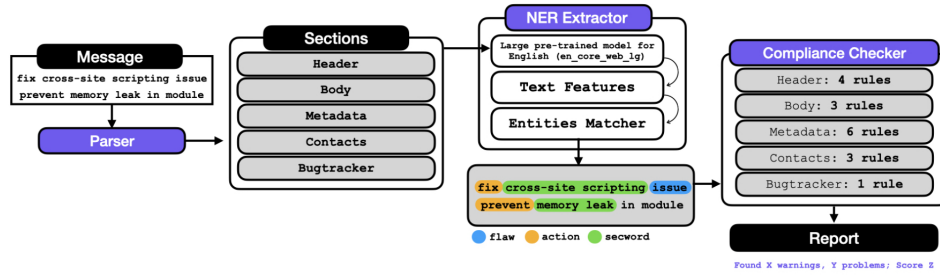


Figure 2.4: Execution steps of SECOMlint and its different components

derived from the application of best practices for generic commit messages, such as references to issues (#NUMBER) (Reis et al., 2023b).

The extractor employs a pre-trained model from spaCy to analyze and extract significant features from the text. Certain features, including part-of-speech tags, are utilized in the NER rules to enhance precision. For example, an action denotes a verb; thus, we extract keywords such as “fix,” “patch,” “prevent,” etc., exclusively when the speech tag is a verb. Following the extraction of various entities for each section, we implement the corresponding sets of section rules. The rules are elaborated upon in greater detail under the GitHub repository¹. The compliance checker considers the types of entities expected to obtain certain rules. For the rule header-ends-with-vuln-id, the header line, which is the first line of the message, is expected to conclude with an entity type of VULNID. The NER extractor is utilized to assist the security engineer in enhancing the body section’s security informativeness. Specifically, the tool offers the option `--is-body-informative`, which verifies the inclusion of any security-related terminology in the body. If not, it requires enhancement for greater clarity. The body section is critical as it provides an explanation of the problem, its significance, and the proposed solution by security engineers (Reis et al., 2023b).

2.4.2 Tools for Automatic Documentation Generation

M. Linares-Vásquez et al. explain in (Linares-Vásquez et al., 2015) that ChangeScribe employs a summarization-based approach, previously presented and evaluated by the developers in their earlier work (Moreno et al., 2014). ChangeScribe extracts and analyzes the differences between two versions of source code and conducts a commit characterization based on the stereotypes of modified, added, and removed methods. The result is a commit message that summarizes the changes and categorizes and elaborates on each modification; it outlines the nature of the change and offers contextual information regarding the rationale in clear language.

Semantic Diff (Jackson and Ladd, 1994) identifies discrepancies between two versions of a procedure and summarizes these semantic differences through program analysis techniques. Additional methods that enhance line-based differencing tools include LDiff (Canfora et al., 2009) and iDiff (Nguyen et al., 2011). Parnin et al. (Parnin and Gorg, 2008) introduced a method for

¹ See GitHub repo 2 at <https://tqrg.github.io/>.

examining variations between program versions at the bytecode statement level. This approach includes the presentation of type information and fully qualified source code locations related to the changes, encompassing both the source entity and the entities affected by the modification. ChangeScribe employs line-based differencing while enhancing the context of changes through a natural language description that incorporates the commit stereotype, change descriptions, and impact set (Linares-Vázquez et al., 2015).

DeltaDoc (Buse and Weimer, 2010a) generates textual descriptions of source code changes automatically through the use of symbolic execution and summarization techniques. When the change-set is extensive, encompassing numerous files or methods, it delineates each method individually, disregarding potential dependencies among them. Rastkar and Murphy introduced a multi-document summarization method aimed at elucidating the rationale for a change. (Rastkar and Murphy, 2013) In contrast to the aforementioned approaches, ChangeScribe-generated commit messages provide more comprehensive information regarding the nature of the changes, including details on dependencies, and do not necessitate the use of multiple artifact types (Linares-Vázquez et al., 2015).

Automatically generating high-quality commit messages is challenging, as explaining the rationale behind a change often necessitates the integration of various types of knowledge and context. Recent studies indicate that commit messages exhibit certain patterns (Jiang and McMillan, 2017; Mockus and Votta, 2000b), and it is feasible to derive patterns of software artifacts from extensive datasets (Hindle et al., 2012). Jiang et al. proposed the adaptation of a neural machine translation (NMT) technique for generating commit messages from change diffs (Jiang et al., 2017). In the remainder of the paper, we will refer to their approach as NMT for the sake of simplicity. NMT seeks to acquire the ability to generate commit messages by analyzing previous changes and their corresponding commit messages. NMT distinguishes itself from previous research by emphasizing the generation of concise messages that elucidate the rationales underlying changes. Jiang et al. presented the performance of their method utilizing a dataset constructed from the top 1,000 Java projects on GitHub (Liu et al., 2018).

NMT offers several advantages over traditional commit message generation methods. First, unlike prior approaches, NMT produces concise summaries instead of verbose descriptions of code changes. It also removes the dependency on manually crafted templates, which tools like DeltaDoc and ChangeScribe require. Furthermore, NMT can generate commit messages for a wide range of software artifact modifications, extending beyond simple source code edits (Liu et al., 2018).

In their investigation, Z. Liu et al. found that effective commit messages often exhibit lexical similarity to one or more code diffs in the training set. Based on this observation, we introduce a more efficient approach—Nearest Neighbor Generator (NNGen)—for automatically producing commit messages from code diffs. NNGen leverages a nearest neighbor algorithm, bypassing the need for model training. To generate a message, it identifies the most similar diff from the training set and reuses its commit message. Experimental results demonstrate that NNGen significantly outperforms NMT on both Jiang et al.’s original dataset and a cleaned version, as measured by

BLEU scores. In addition, NNGen is highly efficient, completing inference on a CPU in 23 to 30 seconds, compared to NMT's 24 to 38 hours on a GPU—making it over 2,600 times faster. A human evaluation further confirms NNGen's superiority, showing statistically significant improvements over NMT (Liu et al., 2018).

Two commonly used evaluation metrics in automatic documentation generation are BLEU and METEOR.

BLEU (Bilingual Evaluation Understudy) is a precision-oriented metric originally developed for machine translation. It compares n-grams in the generated text to one or more reference texts and calculates an overlap score. While widely adopted, BLEU tends to penalize valid paraphrases and is sensitive to exact wording, making it less ideal for tasks where semantic fidelity is more important than surface form.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) addresses some of BLEU's limitations by incorporating stemming, synonym matching, and a recall component. It typically correlates better with human judgment, particularly in tasks where multiple valid phrasings are acceptable. For software documentation tasks, METEOR can offer a more flexible and semantically aware assessment of output quality.

Chapter 3

Empirical Study Design and Dataset

This chapter presents the methodological foundation for the empirical evaluation conducted in this work. It defines the research questions that guide the study and describes the construction of a curated dataset of real-world security patches. The chapter details the data sources, filtering heuristics, and structured schema used to prepare the data for generation and validation tasks. It also provides an overview of the dataset’s statistical characteristics, including patch sizes, CWE distributions, severity levels, and temporal patterns, establishing the basis for the experiments in subsequent chapters.

3.1 Research Questions

This research investigates the use of generative Large Language Models (LLMs) for enhancing the quality and standardization of security patch documentation. In particular, we focus on the generation of commit messages that comply with the Security Commit Message (SECOM) convention. The following research questions guide our empirical evaluation:

Research Question 1 (RQ1). Can generative AI produce SECOM-compliant commit messages for security patches?

The lack of structured and informative security commit messages remains a critical barrier to effective vulnerability detection and patch triage. While the SECOM convention provides a standardized format, producing such messages manually is effort-intensive and error-prone. (Reis et al., 2023b). This research question investigates whether a zero-shot prompting strategy can guide an LLM to generate SECOM-compliant messages based solely on source-level patch data. A sample of 200–400 real-world security patches will be used as input for generation. Each output will be evaluated using SECOMLint, which measures structural compliance and content completeness. We analyze the resulting compliance scores and conduct a qualitative review to identify common failure patterns and practical limitations.

Research Question 2 (RQ2). Do in-context learning approaches improve the quality of generated metadata?

This question investigates whether providing in-context examples during inference improves the ability of the model to generate accurate and complete SECOM-compliant metadata. The motivation arises from observations in the zero-shot setting, where the model was able to generate commit messages that were structurally valid but often lacked key metadata elements or included them incorrectly. In-context learning has been shown to guide large language models toward more accurate task outputs by conditioning generation on relevant examples. This approach is particularly relevant in the context of security commit messages, where fine-tuning is not performed and task-specific training data may be limited.

To address this question, a few-shot prompting strategy is employed using SECOM-compliant commit messages as examples. The performance of the model is then evaluated against the same annotated dataset used in the zero-shot setting, with comparisons made in terms of field coverage, metadata accuracy, and overall format compliance. This setup enables the assessment of whether in-context prompting leads to measurable improvements in metadata generation without altering the underlying model.

3.2 Dataset Overview

To support the evaluation of generative models for the automated generation of security commit messages, we constructed a curated dataset of vulnerability-related commits. The dataset design and construction were guided by methodological rigor, drawing inspiration from the dataset used in the MSR-accepted study Towards Security Commit Message Standardization by Reis et al. (2025). Our objective was to build a high-quality benchmark that enables the assessment of structured message generation under both zero-shot and one-shot prompting conditions.

3.2.1 Data Collection and Sources

All samples included in the dataset were derived from commits that are explicitly linked to known software vulnerabilities, as recorded in publicly maintained and widely recognized vulnerability databases. The primary sources include:

- **Open Source Vulnerabilities (OSV) Database:** A canonical repository of open-source vulnerability metadata, providing structured information such as commit references, CVE and CWE annotations, and publication timelines.
- **National Vulnerability Database (NVD):** Maintained by NIST, the NVD provides enriched metadata for publicly disclosed vulnerabilities, including severity scores, attack vectors, and weakness classifications.

Only commits associated with a valid vulnerability identifier (e.g., CVE or GHSA) and traceable to these databases were retained.

3.2.2 Filtering Heuristics

To ensure dataset quality and suitability for generative modeling, the raw data was subjected to a series of heuristic filters. These filters were motivated by both practical considerations (e.g., computational constraints during inference) and theoretical concerns (e.g., relevance of changes to security patching). The applied heuristics are as follows:

- **Single-File Commits:** Only commits modifying a single source file were retained. This constraint reduces noise from multi-component changes and ensures tractable input lengths for language models, particularly when API-based inference (e.g., OpenAI) or local deployment (e.g., LLaMA via Ollama) is employed.
- **CWE-Annotated Entries Only:** To enable downstream validation of vulnerability classification accuracy, we excluded all commits lacking a valid Common Weakness Enumeration (CWE) identifier. This ensures that each instance is grounded in a known vulnerability class.
- **Exclusion of Non-Code Changes:** Commits that solely affected documentation files (e.g., `.md`, `.txt`, `.rst`), configuration files (e.g., `.toml`), or test directories (`test/`, `tests/`) were filtered out. These changes are unlikely to reflect substantive vulnerability remediation efforts.
- **Presence of Patch Content:** Commits lacking parsable or non-empty patch content were excluded. The patch content field, typically a code diff, serves as the primary input for generative tasks and is therefore a mandatory requirement.

3.2.3 Subset Construction and Management

To facilitate scalable experimentation and prevent data leakage across evaluation conditions, the curated dataset was partitioned into non-overlapping subsets. A dedicated extraction script was developed to automate this process. The subset generation mechanism features the following characteristics:

- **Random Sampling Without Replacement:** Each subset contains a random sample of 100 data points that have not been used in prior subsets. A tracking mechanism ensures that no commit appears in multiple subsets.
- **Reproducible Outputs:** Subsets are serialized in Pandas' table-oriented JSON format, preserving field types and schema conformity.
- **Structured Storage:** Subset files are stored sequentially as `subset_1.json`, `subset_2.json`, etc., in a dedicated directory. The generation process also maintains a sidecar tracking file to manage sampling state.

The subset generator was invoked on the filtered dataset (`secommits_filtered.json`) using the following interface:

```
python extract_subset.py -input dataset/secommits_filtered.json
                        -output-dir subsets
```

This process yielded ten distinct subsets, each comprising 100 vulnerability-related commits. These subsets were used in the zero-shot and few-shot experiments described in Chapter 4 and Chapter 5.

In addition to general subsets, we created targeted thematic subsets to facilitate controlled comparisons across vulnerability types. Specifically, we isolated and grouped commits associated with the most frequently occurring CWE categories related to input validation and injection vulnerabilities. These include:

- **CWE-79:** Improper Neutralization of Input During Web Page Generation (XSS)
- **CWE-82—88:** Variants of script injection and markup misprocessing flaws

Each group was saved in a dedicated file (e.g., `cwe-79_samples.json`, `cwe-88_samples.json`) to support specialized analyses under RQ2. This structuring enabled the evaluation of generative consistency across semantically similar patches and allowed for fine-grained prompt conditioning using vulnerability type.

Reproducibility and Code Availability To support reproducibility, all source code and experimental artifacts developed for this thesis are publicly available in a GitHub repository. A detailed description of its contents and structure is provided in provided in Appendix C.

3.2.4 Data Schema and Fields

Each commit entry in the dataset is a structured JSON object that encapsulates metadata, code-level changes, and security annotations necessary for both generation and evaluation tasks. The following fields are included:

- **message:** The original, human-authored commit message as written by the developer. This message reflects how the developer described the patch and serves as a reference point for evaluating the quality of AI-generated messages.
Example: `fix(reported_content): sanitize report URLs`
- **vuln_id:** A unique vulnerability identifier that links the commit to a known security issue. This ID is typically sourced from databases such as the Open Source Vulnerability (OSV) database or the National Vulnerability Database (NVD).
Example: `GHSA-2xw8-j43j-5vxp`
- **cwe_id:** The Common Weakness Enumeration (CWE) classification describing the nature of the vulnerability. This field contains one or more CWE identifiers.
Example: `{"CWE-79"}` (Cross-site Scripting)

- **score:** The CVSS (Common Vulnerability Scoring System) score that quantifies the severity of the vulnerability on a scale from 0.0 to 10.0.

Example: 5.4

- **patch_content:** The full unified diff of code changes associated with the commit. This field is the primary input for generative models and is formatted in the standard diff format. It includes file paths, line numbers, and changes made to each line.

Example:

```
-$report->address = $address;
+$report->address = elgg_normalize_site_url($address);
```

- **project, commit_sha, commit_datetime:** Metadata fields identifying the source project and the exact commit.

project: <https://github.com/elgg/elgg>

commit_sha: c30b17bf75256ed3fcc84e2083147cc3951423d0

commit_datetime: 12/06/2021, 14:39:10

- **patch:** A string indicating the patch structure, such as "SINGLE" for single-commit patches.
- **chain, chain_len, chain_ord:** Fields that capture the structure of the commit chain used to implement the fix.
 - chain:* URLs of all commits involved in the fix.
 - chain_len:* Total number of commits in the patch chain.
 - chain_ord:* Ordered list of commit SHAs showing the fix sequence.
- **before_first_fix_commit, last_fix_commit:** SHAs marking the state before the patch and the final fix commit.
- **code_diff:** A simplified form of `patch_content`, showing only the diff section (omitting headers). Useful for parsing small edits.

- **author:** The original author of the patch commit.

Example: Jérôme Bakker

- **files, list_files, num_files:** File-level metadata.

files: A dictionary detailing the changes in each file (additions, deletions, status, raw URL, and per-file diffs).

list_files: Extracted list of modified files.

num_files: Number of files affected in the commit.

- **language:** The detected language of the message. Used for filtering multilingual entries.

Example: ro

- **entities**: Named entities extracted using NER rules, typically used in the SECOMLint evaluation process.

Example: [('fix(reported_content', 'ACTION', '"), ('sanitize', 'SECWORD', '")]

- **comments (optional)**: External comments from PRs or issue discussions, if available.

3.2.5 Dataset Statistics and Characteristics

Patch Size and Commit Distribution To understand how security patches are applied in practice, the dataset was analyzed for its `chain_len` field, which describes how many commits are associated with each patch. The results show that:

- 89.4% of vulnerabilities are patched using a single commit.
- The remaining 10.6% require multiple commits.

This distribution is illustrated in Figure 3.1.

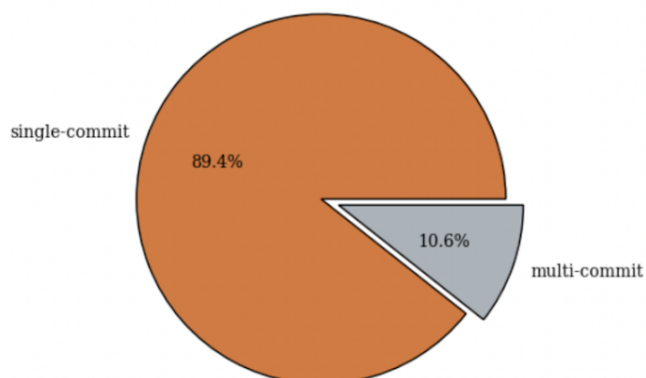


Figure 3.1: Pie chart showing distribution of single vs multi-commit patches

In addition, the size of each patch was measured based on its structural footprint. A log-scale bar chart was used to illustrate the skewed distribution. This is shown in Figure 3.2.

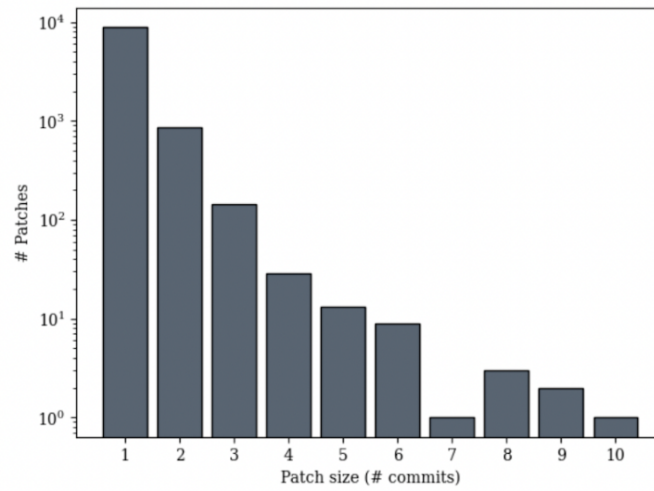


Figure 3.2: Bar chart showing the distribution of patch sizes.

Severity Levels and Vulnerability Types The `cvss_score` field was used to categorize vulnerabilities by severity, following industry-standard brackets:

- Low (0.1—3.9)
- Medium (4.0—6.9)
- High (7.0—8.9)
- Critical (9.0—10.0)

This severity distribution is illustrated in Figure 3.3.

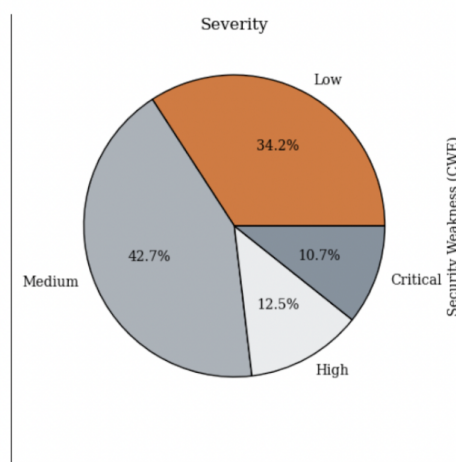


Figure 3.3: Pie chart showing distribution of vulnerabilities by CVSS severity level

To complement this, the top 15 most frequently occurring CWE categories were extracted and visualized. The most common types included those listed in Table 3.1.

Table 3.1: Top 15 CWE Categories in the Dataset

CWE ID	Title	Description
CWE-79	Cross-site Scripting (XSS)	Improper neutralization of input during web page generation, leading to script execution in a user's browser.
CWE-125	Out-of-bounds Read	Accessing memory outside the intended buffer, potentially causing crashes or information leaks.
CWE-20	Improper Input Validation	Failure to properly validate input, allowing malformed data to influence program behavior.
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	Inadequate enforcement of memory bounds, leading to overflows or memory corruption.
CWE-787	Out-of-bounds Write	Writing outside the bounds of allocated memory, potentially leading to crashes or code execution.
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	Unintended disclosure of sensitive data via logs, error messages, or insecure APIs.
CWE-476	NULL Pointer Dereference	Attempting to access memory via a null pointer, typically resulting in crashes.
CWE-22	Path Traversal	Inadequate control over file paths, enabling access to unauthorized files or directories.
CWE-400	Uncontrolled Resource Consumption	Operations that consume excessive resources, potentially leading to denial of service.
CWE-89	SQL Injection	Failure to neutralize user input in SQL queries, allowing manipulation of database operations.
CWE-416	Use After Free	Accessing memory after it has been released, often leading to crashes or exploitable behavior.
NVD-CWE-noinfo	No Information Available	No detailed CWE classification provided in the corresponding advisory.
CWE-190	Integer Overflow or Wraparound	Arithmetic operations that exceed variable bounds, possibly affecting logic or memory.
NVD-CWE-Other	Other / Miscellaneous	The vulnerability is categorized under unspecified or miscellaneous CWE types.
CWE-352	Cross-Site Request Forgery (CSRF)	An attacker tricks an authenticated user into executing unwanted actions.

This distribution is shown in Figure 3.4.

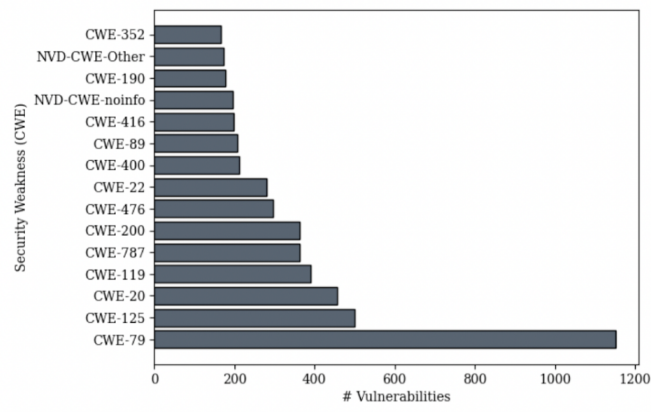


Figure 3.4: Top 15 most frequent CWE categories in the dataset

Temporal Patterns in Patching To investigate longitudinal trends in security patching, we analyzed the `commit_datetime` associated with each vulnerability-related commit, which captures when a patch was applied at the source-code level.

The temporal distribution shown in Figure 3.5 reveals a consistent increase in patching activity between 2010 and 2022, with a notable surge from 2018 onward. This pattern reflects growing adoption of secure development practices, likely driven by the rise of coordinated vulnerability disclosure, the integration of security automation tools, and stricter compliance standards. The peak in recent years suggests a heightened focus on timely remediation and transparent patch delivery across software ecosystems.

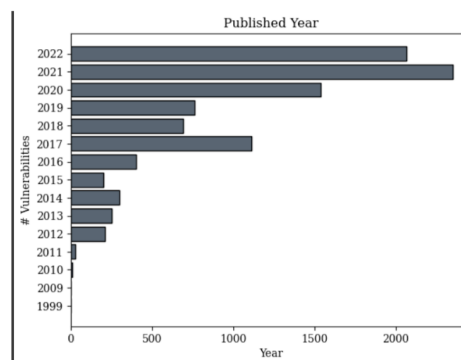


Figure 3.5: Number of security patches over time (2010—2022)

Chapter 4

Zero-Shot Generation of SECOM-Compliant Security Commit Messages

This chapter addresses RQ1 through an initial attempt to discover whether large language models can generate SECOM-compliant security commit messages in a zero-shot setting. It outlines the experimental setup, including the prompting strategy, model configuration, and generation pipeline. The chapter then evaluates the outputs using SECOMLint for structural compliance and compares generated metadata fields—such as CWE ID, severity, and CVSS score—against ground truth annotations. The findings reveal that, while the model demonstrates strong adherence to format, it struggles with metadata accuracy, highlighting the limitations of unguided generation in this context.

4.1 Model and Prompting Configuration

This study investigates the capacity of large language models (LLMs) to generate standardized, SECOM-compliant security commit messages using a zero-shot prompting approach. To this end, we employed OpenAI’s `gpt-4.1-mini` model [OpenAI \(2024\)](#), accessed through the OpenAI Chat API. This section provides an overview of the model selection, the design of the prompting strategy, and the underlying rationale behind the methodology.

GPT and the `gpt-4.1-mini` Model

Generative Pre-trained Transformers (GPT) [OpenAI \(2024\)](#) are autoregressive language models trained on large-scale text corpora to predict the next token in a sequence. These models have demonstrated strong performance across a wide range of natural language processing tasks, often with minimal task-specific supervision.

For this work, we used `gpt-4.1-mini` OpenAI (2024), a lightweight variant of OpenAI’s GPT-4 model family. Despite its reduced size, `gpt-4.1-mini` supports multi-turn conversational contexts and instruction-following capabilities, making it suitable for zero-shot generation tasks framed using carefully engineered prompts.

The model was accessed programmatically via the OpenAI Python SDK, with the following generation parameters:

- `temperature = 0.0` (ensures deterministic output)
- `max_tokens = 1000` (allows full SECOM message generation)
- `top_p = 1.0, frequency_penalty = 0.0, presence_penalty = 0.0`

Prompting Strategy

The task was framed as a zero-shot generation problem, meaning that the model was not provided with any in-context examples. Instead, the desired structure and constraints were specified exclusively through natural language instructions. This setup utilized the two-role format supported by OpenAI’s Chat API:

- A **system prompt**, which defines the model’s role and the structure of the expected output.
- A **user prompt**, which provides the code diff and initiates the generation task.

In this setup, all task-specific instructions—including formatting guidelines, stylistic constraints, and behavioural expectations—are embedded within the system prompt. The user prompt serves a purely minimal role: it introduces the code diff and triggers the generation task. This design maintains a strict zero-shot setting and evaluates the model’s ability to generalize from a single, well-structured instruction set.

System Prompt The system message establishes the role of the model as an “expert software engineer generating standardized security commit messages according to the SECOM convention.” It outlines the commit message format in detail, including:

- A required `vuln-fix:` prefix in the header line.
- A vulnerability ID enclosed in parentheses.
- A `<Body>` section consisting of exactly three sentences:
 - Sentence 1: *What* the vulnerability is.
 - Sentence 2: *Why* it is a security risk.
 - Sentence 3: *How* it was fixed.
- Structured metadata fields (e.g., CWE-ID, CVSS score, detection method, report URL).

A full version of the system prompt used in this configuration, including all structural fields and formatting constraints, is included in Appendix B.

The prompt explicitly forbids paragraph-style bodies, requiring each sentence to start on a new line. A concrete example of a valid body section is also included in the prompt to illustrate the expected structure and tone. This prompt design was iteratively refined to balance syntactic rigidity (for SECOMLint compatibility) with linguistic clarity and informativeness.

User Prompt The user prompt was deliberately kept minimal to reflect a pure zero-shot setting. Its function was to pass the patch content as-is to the model:

```
Generate a SECOM-style security commit message for the following code diff:
```

```
```diff
<code_diff>
```

The model must derive all relevant information—vulnerability type, context, severity, and fix rationale—exclusively from the code diff. This setup tests the model’s generalization capabilities and mirrors realistic automation scenarios, in which developers or tools might request commit messages with minimal overhead.

This combined prompting strategy—composed of a high-information system role and a minimalist user instruction—forms the core of our zero-shot methodology. It sets the stage for evaluating the model’s ability to generate structured and meaningful security patch documentation across a diverse set of real-world vulnerabilities.

## 4.2 Data Processing and Message Generation Pipeline

To evaluate the zero-shot generation capabilities of GPT-4.1-mini for SECOM-compliant commit messages, we implemented an automated inference pipeline in Python. This subsection outlines the data structure, execution flow, and generation logic used in our experiments.

### Dataset and Structure

The dataset comprises structured JSON files, each representing a filtered and curated subset of 100 vulnerability-related commits. We used five such subsets (500 entries total), selected according to filtering heuristics described in Section 3.2.2. Each entry corresponds to a single commit addressing a known vulnerability and includes both metadata and patch content.

### Dataset Entry Structure

Each record in the dataset corresponds to a single security-relevant commit and is represented as a structured JSON object. The data captures not only the patch content but also detailed contextual

and semantic metadata associated with the vulnerability and the commit. Below is an overview of the fields contained in a typical data entry:

- `index`: Internal identifier for the entry.
- `vuln_id`: The vulnerability identifier (e.g., CVE or GHSA code), such as `GHSA-6mv9-hcx5-7mhh`.
- `cwe_id`: The CWE classification associated with the vulnerability (e.g., `CWE-918`).
- `score`: The CVSS severity score, e.g., 5.3.
- `summary`: A textual description of the vulnerability, as reported in public advisories.
- `project`, `commit_href`, `commit_sha`, `author`, `commit_datetime`: Metadata identifying the source project, commit location, author, and timestamp.
- `patch_content`: The full commit diff, including file metadata and code changes.
- `code_diff`: A simplified diff fragment extracted from `patch_content`, used as the input to the language model.
- `message`: The original commit message authored by the developer.
- `message_norm`: A normalized lowercase version of the commit message for downstream analysis.
- `chain`, `chain_ord`, `chain_len`, `last_fix_commit`: Information about the patch chain and its position in the remediation sequence.
- `files`, `list_files`, `num_files`: File-level metadata including changes per file and total file count.
- `entities`: Named entities extracted from the message using a lightweight rule-based NER system (e.g., security keywords, verbs, etc.).

The inclusion of both human-written messages and code diffs allows the dataset to serve dual purposes: it supports both prompt-based generation and reference-based evaluation. The extracted `code_diff` field is used as the sole input to the language model in our zero-shot prompting setup, while fields such as `vuln_id`, `cwe_id`, and `message` are preserved for evaluation and metadata tracking.

An abbreviated version of a representative entry is included below:

```

1 {
2 "vuln_id": "GHSA-6mv9-hcx5-7mhh",
3 "cwe_id": "'CWE-918'",
4 "score": 5.3,
5 "summary": "... allows an attacker to have Jenkins submit HTTP GET
6 requests...",
7 "commit_sha": "2d16b459205730d85e51499c2457109b234ca9d9",
8 "commit_datetime": "01/30/2018, 17:15:48",
9 "message": "[SECURITY-506] Require admin permission to validate proxy
10 config.",
11 "author": "Jesse Glick",
12 "code_diff": "@@ -341,6 +341,8 @@ public FormValidation
 doValidateProxy(...) {\n+
 Jenkins.getInstance().checkPermission(Jenkins.ADMINISTER);\n+ ..."}

```

Listing 4.1: Representative JSON entry used in evaluation.

This rich structure enables fine-grained analysis of both the model’s input conditions and its outputs.

### Zero-Shot Generation Procedure

For each commit in the dataset, we prompt the model to generate a full SECOM-style security commit message using a pure zero-shot configuration. The pipeline proceeds as follows:

1. Load the subset into a pandas DataFrame.
2. Extract the `code_diff` field as input and the `vuln_id`, `cwe_id`, and `message` for meta-data.
3. Construct a prompt using the system-user format supported by OpenAI’s `chat.completions` endpoint:
  - The **system message** encodes the full SECOM format, output constraints, and stylistic rules (see Section 4.1).
  - The **user message** consists of a short instruction and the code diff enclosed in a `diff` code block.
4. Send the prompt to the `gpt-4.1-mini` model with deterministic decoding parameters (`temperature = 0.0`).
5. Extract the returned commit message, strip any formatting artifacts (e.g., markdown fences), and store the result along with the original metadata.

This procedure was repeated across five subsets, each comprising 100 commits, resulting in a total of 500 generated commit messages. The results were serialized to CSV files for further validation and scoring.

### Reproducibility and Automation

The pipeline was designed for full automation and reproducibility. A single script performs all operations end-to-end, from reading and processing input data to executing inference and saving the results. This ensures consistency across subsets and allows for efficient scaling to additional data.

By restricting the user prompt to only the code diff and deferring all structural and formatting instructions to the system message, this setup evaluates the model’s ability to generate standardized documentation from raw code changes in a strictly zero-shot setting.

## 4.3 Validation Strategy and Results Analysis

This section outlines the methodology used to evaluate the outputs generated under the zero-shot prompting setup. It details the metrics employed to assess structural conformity, semantic accuracy, and metadata fidelity of the generated commit messages, followed by an analysis of the observed results.

### 4.3.1 Validation Strategy

To assess the structural and semantic compliance of the generated commit messages, we employed a two-fold validation approach. The first stage of evaluation relied on **SECOMLint** [Reis et al. \(2023b\)](#), an automated linter designed specifically to assess adherence to the SECOM commit message convention. This tool was used to compute compliance scores for both the original human-authored messages and those generated by the language model. The second stage of validation, discussed in the next subsection, focuses on metadata alignment and content plausibility.

#### SECOMLint: Rule-Based Compliance Evaluation

SECOMLint evaluates commit messages by checking whether they satisfy a set of syntactic and semantic constraints defined by the SECOM standard. Each rule in SECOMLint corresponds to a specific formatting or content requirement. Rules can be configured with three parameters:

- **active**: Indicates whether the rule is enabled (1) or disabled (0).
- **type**: Defines whether the rule violation is classified as a warning (0) or an error (1).
- **value**: Specifies the threshold or expected pattern, often derived from Named Entity Recognition (NER) over the commit message.

SECOMLint divides a commit message into five logical sections: `header`, `body`, `metadata`, `contact`, and `bug-tracker references`. Each section is checked for completeness, consistency, and adherence to formatting guidelines.

Table 4.1 summarizes the full set of SECOMLint rules used in our validation, along with their purpose and severity classification.

Table 4.1: SECOMLint rules and corresponding validation criteria

Rule Name	Condition / Description	Severity	Value
header-max-length	Header must not exceed specified number of characters.	Warning	50
header-is-not-empty	Header must be non-empty and contain more than 0 characters.	Error	0
header-starts-with-type	Header must begin with the specified type prefix.	Error	vuln-fix
header-ends-with-vuln-id	Header should end with a vulnerability ID.	Warning	VULNID
body-is-not-empty	Body section must be present and contain more than 0 characters.	Error	0
body-max-length	Body should not exceed the specified number of characters.	Warning	75
body-has-three-paragraphs	Body must contain exactly three paragraphs.	Error	3
metadata-has-weakness	Metadata must include a <code>weakness:</code> tag followed by CWE-ID.	Error	CWEID
metadata-has-severity	Metadata must include a <code>severity:</code> field.	Error	SEVERITY
metadata-has-detection	Metadata should include a <code>detection:</code> method.	Warning	DETECTION
metadata-has-report	Metadata should contain a <code>report:</code> URL.	Warning	URL
metadata-has-cvss	Metadata should include a <code>cvss:</code> score.	Warning	-
metadata-has-introduced-in	Metadata should specify the commit SHA where vulnerability was introduced.	Warning	SHA
contact-has-reported-by	Contact section should list a <code>reported-by:</code> name and email.	Warning	EMAIL
contact-has-signed-off-by	Contact section should list a <code>signed-off-by:</code> name and email.	Warning	EMAIL
contact-co-authored-by	Contact section may include a <code>co-authored-by:</code> field.	Warning	EMAIL
bugtracker-has-reference	Bug tracker section should include a reference using <code>bug-tracker:</code> , <code>resolves:</code> , or <code>see also:</code> .	Warning	URL/ISSUE

Each of the SECOMLint rules listed in Table 4.1 is individually evaluated for compliance. The output of this validation process is a binary result for each rule: either the generated message satisfies the rule (compliant), or it does not (non-compliant). To compute a single, interpretable score that reflects the structural quality of a commit message, SECOMLint aggregates the rule outcomes using a simple average.

Formally, the compliance score is calculated as the ratio between the number of rules passed and the total number of active rules. If a message satisfies 13 out of 17 applicable rules, the resulting score is  $13/17 = 0.76$ . This score is normalized to fall within the range  $[0, 1]$ , where 1.0

indicates full structural compliance with the SECOM convention.

The implementation of this computation follows a straightforward logic: it iterates over all rule results, counts how many rules were satisfied, and divides this count by the total number of rules. This approach ensures that the final score reflects overall structural adherence, rather than penalizing any single field disproportionately. It also enables comparative evaluation across different messages and prompting strategies.

Together, these rules form the compliance framework used to assess the quality of security commit messages. Violations of `type=1` (errors) are considered critical and indicate a failure to meet SECOM standards. `type=0` rules (warnings) are more lenient but still contribute to overall quality degradation. By comparing rule violations between human and generated messages, we quantify the structural fidelity of the model's outputs.

SECOMLint was integrated directly into the evaluation pipeline as a post-processing step. The tool operates over structured CSV files and is designed to analyze batches of commit messages at once. Given a CSV input, SECOMLint automatically identifies all columns whose names end with `_message` (e.g., `original_message`, `generated_secom_message`) and applies its rule set to each message in those columns.

For every input message, SECOMLint computes:

- A rule-by-rule compliance score, indicating whether the message satisfies each enabled constraint.
- A global score summary that aggregates the number of errors and warnings across all rules.
- A structured list of named entities extracted from the message, including actions, CWE identifiers, severity levels, CVSS scores, URLs, and contact names or emails.

The tool appends these results as new columns in the output CSV file, preserving the original messages and metadata while enriching them with validation outcomes. This design allows for efficient comparative analysis between the original (human-written) and generated commit messages.

In our case, SECOMLint was used to evaluate two columns:

- `original_message`: the human-authored commit message from the original dataset.
- `generated_secom_message`: the message produced by the language model under zero-shot prompting.

By comparing the SECOMLint scores for these two columns, we gain insights into the structural completeness, semantic richness, and formatting fidelity of generated messages relative to human-authored ones.

### Metadata Consistency Validation

To complement structural evaluation, we implemented a secondary validation strategy focused on semantic accuracy. This validation stage checks whether key metadata values present in the

generated SECOM messages align with their corresponding ground-truth values from the original dataset.

A custom Python script was used to automate this process. For each entry, the following fields were validated:

- **CWE ID (Weakness):** A regular expression was used to extract the first occurrence of a CWE tag (e.g., `CWE-79`) from the generated message. The extracted value was normalized and compared against the original `cwe_id` field.
- **Severity Level:** The CVSS-based severity (`low`, `medium`, `high`, `critical`) was computed from the original CVSS score and compared against a severity label found in the generated message.
- **CVSS Score:** A numeric CVSS value was parsed from the generated message (if present), and compared to the original score. A match was recorded if the difference was less than 0.1.

Each comparison was encoded as a boolean flag (`match = True/False`). The output of this validation step was saved to a CSV file for analysis, with each row containing the original metadata, the values extracted from the generated message, and a per-field match indicator.

The validation was applied to all five subsets used during generation. A final merged file, `secom_zero_shot_metadata_validation.csv`, was produced summarizing the consistency results across 500 generated messages. This enabled quantitative analysis of how faithfully the model was able to infer and reproduce ground-truth metadata in a zero-shot setting.

It is important to note that certain SECOM metadata fields—specifically `reported-by`, `signed-off-by`, `co-authored-by`, `report`, and `see also`—were excluded from this validation process. These fields do not exist in the original dataset and therefore lack a reliable ground-truth reference for comparison. As such, their presence or absence in the generated messages was not evaluated.

Second, the generated messages were evaluated against their corresponding human-written references using standard text similarity metrics. Specifically, **average BLEU** and **average METEOR** scores were computed using custom Python scripts. BLEU (Bilingual Evaluation Understudy) is a widely used metric in machine translation and text generation that measures how many words and short word sequences (n-grams) in the generated output match those in a reference sentence. In this context, it reflects how well the model’s output captures the phrasing and structure of the original commit message. The BLEU score was calculated using the `sentence_bleu` function from the NLTK library with smoothing to handle cases where n-gram matches are sparse.

METEOR (Metric for Evaluation of Translation with Explicit ORdering), on the other hand, is designed to account for linguistic variations more flexibly. It considers not only exact word matches but also stems, synonyms, and word order, making it more sensitive to semantic similarity than BLEU. In this study, METEOR scores were computed using the `meteor_score` function on tokenized inputs. Both metrics were calculated for each generated message individually and then

averaged over the full evaluation set. Both BLEU and METEOR scores range from 0 to 1, with higher values indicating better similarity between the generated message and its human-written counterpart. A score of 1.0 represents a perfect match. BLEU is more conservative, rewarding exact n-gram overlaps, while METEOR incorporates synonymy, stemming, and reordering, making it more sensitive to meaning even when wording varies.

#### 4.3.2 Results: SECOMLint Compliance

The first evaluation metric analyzed was the structural compliance score produced by SECOMLint. For each message in the dataset, SECOMLint outputs a scalar score reflecting the proportion of satisfied rules, normalized over the full rule set.

To quantify overall adherence, we computed the average compliance scores for both the original (human-authored) commit messages and those generated via the zero-shot prompting setup. These values were obtained by aggregating rule-level results across all five subsets used in the experiment and are summarized in Table 4.2.

Table 4.2: Average compliance scores for original and generated messages

Message Type	Average Compliance Score
Original (human-authored)	0.1390
Generated (zero-shot prompting)	0.6221

Table 4.3: Example of original vs. generated commit message and SECOMLint scores

<b>Patch Diff (excerpt)</b>		<code>QuantizeAndDequantizeV4GradientOp : public OpKernel { [...] + OP_REQUIRES(ctx, TensorShapeUtils::IsScalar(input_min_tensor.shape()), [...] }</code>
<b>Original Message</b>		<code>Fix tf.raw_ops.QuantizeAndDequantizeV4Grad vulnerability with invalid input_min or input_max. Check that argument is actually a scalar before treating it as such. PiperOrigin-RevId: 445198280</code>
<b>Generated Message</b>		vuln-fix: Enforce scalar input validation for min/max tensors (CVE-2024-12345)   The vulnerability allowed non-scalar input_min and input_max tensors when axis was unspecified, leading to potential out-of-bounds memory access or incorrect gradient computations.   This posed a security risk by enabling attackers to cause crashes or corrupt memory through malformed tensor shapes.   The fix enforces strict scalar shape validation for input_min and input_max tensors when axis is -1, preventing invalid tensor shapes from being processed.   <b>Weakness:</b> CWE-125: Out-of-bounds Read <b>Severity:</b> High <b>CVSS:</b> 7.5 <b>Detection:</b> Code review and fuzz testing <b>Report:</b> <a href="https://security-advisories.example.com/CVE-2024-12345">https://security-advisories.example.com/CVE-2024-12345</a> <b>Introduced-in:</b> abcdef1234567890abcdef1234567890abcdef12   <b>Reported-by:</b> Security Researcher (security@example.com) <b>Reviewed-by:</b> Jane Doe (jane.doe@example.com) <b>Signed-off-by:</b> John Smith (john.smith@example.com)   <b>Bug-tracker:</b> <a href="https://bugs.example.com/issue/1234">https://bugs.example.com/issue/1234</a> <b>Resolves:</b> #1234 <b>See also:</b> #1233
<b>SECOMLint</b>	<b>Score</b>	0.12
<b>SECOMLint</b>	<b>Score</b>	0.65
<b>(Original)</b>		
<b>(Generated)</b>		

The original message is terse and lacks structure, omitting key vulnerability descriptors, CWE

classifications, severity information, and standard formatting. In contrast, the generated message strictly follows the SECOM convention, provides a well-structured narrative (what, why, how), and includes the required metadata fields.

This example is representative of the broader trend observed in the dataset: while original messages often contain minimal information and non-standard phrasing, the generated outputs consistently exhibit higher conformity and documentation quality.

The results indicate a substantial increase in SECOM compliance among generated messages—representing a **+383% relative improvement** over the original commit messages. This suggests that, even in the absence of in-context examples, the language model was able to internalize and reproduce the SECOM specification when guided by a well-crafted system prompt.

The high average score for generated messages also indicates that most outputs satisfied the core structural rules of the SECOM format, including presence of a properly formatted header, three-sentence body structure, and inclusion of metadata fields. By contrast, the low score for original messages highlights the widespread inconsistency and lack of standardization in real-world commit documentation for security patches.

#### 4.3.3 Results: Metadata Accuracy

The second evaluation metric examined the factual consistency of critical metadata fields extracted from the generated SECOM messages. Specifically, we evaluated whether the language model was able to accurately reproduce the `CWE ID`, `severity level`, and `CVSS score` associated with each vulnerability, as defined in the original dataset.

Figures 4.1, 4.2, and 4.3 summarize the results using pie charts that show the percentage of exact matches for each metadata field across 500 generated messages.

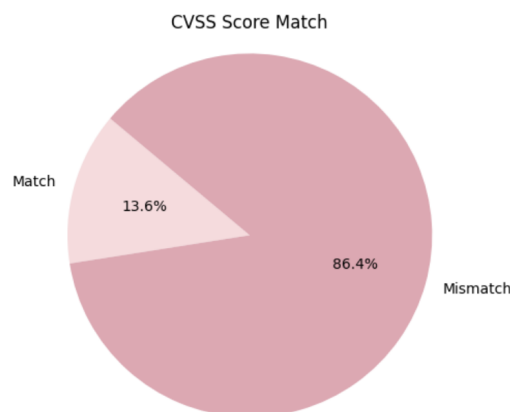


Figure 4.1: CVSS score match between generated message and ground truth.

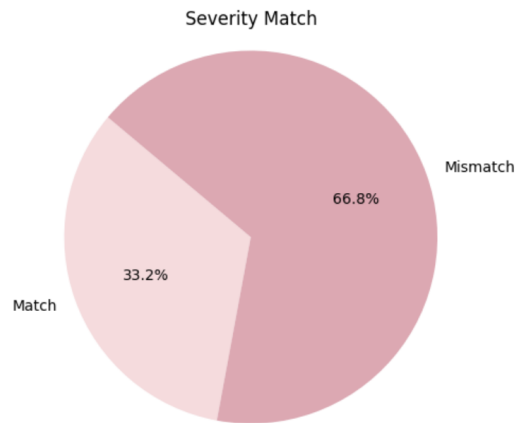


Figure 4.2: Severity level match between generated message and ground truth.

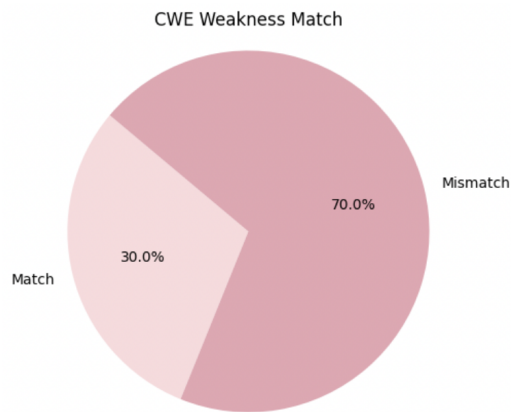


Figure 4.3: CWE weakness match between generated message and ground truth.

The results indicate that while the generated messages exhibited strong structural conformity, their accuracy in reproducing exact metadata values was considerably lower. Only **13.6%** of messages correctly reproduced the original CVSS score (within a  $\pm 0.1$  margin), while **33.2%** accurately matched the severity level. The match rate for CWE identifiers was similarly limited, at **30.0%**.

These findings suggest that although the language model was often able to infer the general structure and intent of a patch, it struggled to extract or replicate fine-grained details such as numeric risk assessments or standardized vulnerability classifications. This has direct implications for the practical reliability of zero-shot generated documentation in security-critical settings.

The similarity-based evaluation using BLEU and METEOR provides an additional perspective on how closely the generated messages align with human-authored ones. As shown in Table 4.4, the generated messages achieved a BLEU score of 0.0036 and a METEOR score of 0.1205, both of which exceed the corresponding scores for the original messages when evaluated in reverse (0.0010 BLEU and 0.0346 METEOR).

Although the absolute values of these metrics are low—consistent with the known limitations of using BLEU and METEOR on short, structurally diverse texts—the relative improvement in the generated messages suggests that they capture more semantic and structural similarity to the originals than vice versa. The higher METEOR score, in particular, reflects the model’s ability to produce text that aligns better with the intent and content of the reference messages, even when exact phrasing differs.

These results support the hypothesis that large language models, even in zero-shot settings, can generate commit messages that are more aligned with structured security documentation conventions than those written by humans under typical development conditions.

Table 4.4: Average BLEU and METEOR scores for generated and original messages

Message Type	BLEU Score	METEOR Score
Generated Message	0.0036	0.1205
Original Message	0.0010	0.0346

#### 4.3.3.1 Discussion: Implications and Limitations. Threats to Validity

The experimental results highlight a key trade-off in using zero-shot prompting for structured documentation tasks. On one hand, the generated commit messages significantly outperformed the original human-written messages in terms of formal structure and SECOM compliance. This indicates that large language models are effective at capturing syntactic patterns and stylistic conventions when guided by clear textual instructions.

Textual similarity metrics further reinforce this observation. As shown in Table 4.4, the generated messages achieved notably higher BLEU and METEOR scores compared to the original messages when evaluated in both directions, with a relative improvement of **260%** in BLEU (0.0036 vs. 0.0010) and **248%** in METEOR (0.1205 vs. 0.0346). Although the absolute values of these metrics remain low—reflecting the inherent diversity and brevity of commit messages—the relative gains suggest that the generated content more closely adheres to expected structure and semantics.

However, these strengths are counterbalanced by a critical limitation: the inability of the model to consistently extract or infer precise factual metadata. The metadata validation analysis revealed that only **13.6%** of generated messages correctly reproduced the original CVSS score, **33.2%** matched the severity level, and **30.0%** matched the CWE identifier. These results highlight that, while the model is adept at producing syntactically valid and contextually plausible text, it struggles with reliably replicating structured vulnerability metadata from source code changes alone.

This gap in factual accuracy significantly undermines the practical utility of zero-shot generated messages in security-critical workflows, where incorrect or missing metadata can compromise downstream tasks such as vulnerability triage, prioritization, and audit logging.

These findings collectively imply that **zero-shot prompting is insufficient when metadata fidelity is essential**. To address this limitation, the subsequent chapter explores whether in-context learning strategies—where the model is primed with curated examples of SECOM-compliant messages—can improve the accuracy and consistency of generated metadata. Specifically, the next set of experiments evaluates the model’s ability to produce correct CWE identifiers, severity levels, and CVSS scores when provided with few-shot demonstrations.

These findings directly motivate **Research Question 2 (RQ2)**, which investigates the effectiveness of example-driven prompting in improving metadata fidelity. By transitioning from a purely zero-shot setup to one that incorporates carefully constructed exemplars, we hypothesize that the model can better align its outputs with ground-truth data, thereby narrowing the accuracy gap observed in this phase of the evaluation.

Ultimately, while zero-shot prompting demonstrates clear benefits in terms of structural standardization, the results reveal its limitations for factual accuracy. This underscores the need for more advanced prompting strategies and enriched context provisioning in the domain of security documentation.

### Threats to Validity

The findings presented in the zero-shot evaluation are subject to several validity threats. First, the zero-shot prompting strategy relies on a single, static instruction template. While this simplifies reproducibility, it limits the generality of the results: alternative formulations of the prompt may have yielded different outcomes. The prompt was not tuned for optimality, nor adapted per CWE category, which may have constrained the model’s ability to generate accurate or context-aware metadata.

Second, the evaluation assumes that the language model has no prior exposure to the examples in the test set. While the dataset was filtered to exclude explicitly seen CVE identifiers, the underlying model may still retain latent knowledge of similar commits or codebases due to its large-scale pretraining. This introduces a risk of knowledge leakage that could inflate performance metrics, particularly for common vulnerability patterns.

Third, structural quality was assessed using SECOMLint, a custom rule-based validator. Although SECOMLint enforces a strict and interpretable compliance regime, it may suffer from implementation flaws, incomplete rule coverage, or overly rigid thresholds that fail to account for contextually acceptable variation. These limitations could result in false negatives (i.e., penalizing structurally valid outputs) or false positives.

Fourth, semantic fidelity was measured using exact-match comparisons for metadata fields (`cvss`, `severity`, and `weakness`). This binary approach does not capture near-correct outputs or cases where the model generates plausible but imprecise values. As such, the metadata accuracy results may understate the model’s ability to approximate real-world metadata under uncertainty.

Finally, all experiments were performed using a single version of a general-purpose language model. The results may not extend to domain-adapted variants, code-specialized models, or models configured with different decoding parameters. Broader conclusions about zero-shot generation for security documentation should be validated across model families and task settings.

## Chapter 5

# Improving SECOM Metadata Accuracy with Few-Shot Learning

This chapter addresses RQ2 by investigating whether few-shot prompting can overcome the metadata limitations observed in the zero-shot setup. It begins with a feasibility analysis of SECOM fields to determine which components are realistically generatable. The chapter then introduces an in-context learning pipeline using curated examples from multiple CWE categories and evaluates the impact on metadata fidelity, structural compliance, and severity estimation. The results show that few-shot prompting leads to meaningful improvements in both metadata accuracy and adherence to the SECOM format, particularly for well-represented vulnerability classes.

### 5.1 Motivation and Metadata Focus for RQ2

The results obtained from the zero-shot generation experiments in Chapter 4 revealed a clear pattern: while structural and stylistic compliance with the SECOM format improved significantly, the accuracy of specific metadata fields—such as `cwe_id`, severity, and CVSS score—remained poor. In particular, over 85% of generated messages failed to reproduce the correct CVSS value, and only about 30% accurately matched the associated weakness.

These limitations suggest that the language model, when operating in a zero-shot setting, struggles to infer factual metadata solely from the code diff. As a result, it becomes necessary to assess which SECOM fields are *feasibly learnable* by a model in practice and which ones require explicit conditioning. This forms the foundation of Research Question 2 (RQ2), which investigates whether incorporating structured context improves metadata fidelity.

To guide this feasibility analysis, we begin by extending the structural breakdown of the SECOM format with a generability classification. Table 5.1 presents the full list of SECOM fields, grouped by functional component, and annotated with their feasibility of generation and necessary retrieval strategies. This enriched classification clarifies which fields can be reliably produced by a language model, which require external validation, and which are entirely dependent on external

data sources. It also helps delimit the experimental scope and motivates our targeted focus on metadata fields in the subsequent evaluations.

Table 5.1: SECOM format components grouped by category, with generability and retrieval comments.

Category	SECOM Field	Generability	Comments
HEADER	vuln-fix	Generatable	Static prefix, deterministically insertable from prompt.
	header	Generatable	Can be reliably generated using imperative prompting.
	vuln_id	Not generatable	Must be retrieved from CVE/GHSA databases or advisory sources.
BODY	body	Generatable	Follows a "what-why-how" structure; consistently produced with prompt templates.
METADATA	weakness	Generatable with validation	CWE may be inferred but must be validated against official taxonomy.
	severity	Generatable with validation	Roughly inferable, but should be mapped to standardized CVSS levels.
	cvss	Generatable with validation	Approximate values can be predicted; precise scores need CVSS calculator or advisory lookup.
	detection	Not generatable	Requires contextual knowledge; must be sourced from disclosure or test reports.
	report	Not generatable	URLs to advisories must be retrieved from trusted security sources.
CONTACTS	introduced-in	Not generatable	Requires version control analysis, e.g., <code>git blame</code> or issue history inspection.
	reported-by	Maybe generatable	Can be included if explicitly present in trailers; otherwise from advisory metadata.
	reviewed-by	Maybe generatable	Pull request metadata or structured trailers required.
	co-authored-by	Maybe generatable	Must be present in commit metadata; otherwise needs manual addition.
	signed-off-by	Maybe generatable	Often included in trailers; not reliably inferred if absent.
REFERENCES	bug-tracker/issues	Maybe generatable	Referenced issues may appear in messages; otherwise require API-based retrieval.

Category	SECOM Field	Generability	Comments
	resolves	Maybe generatable	May be stated directly; else sourced from PR or issue metadata.
	see also	Maybe generatable	Dependent on presence of explicit references; may need manual review.

## Feasibility Analysis of SECOM Field Generation

This section examines the feasibility of generating each component of a SECOM-compliant commit message using language models. Table 5.1 presents the full set of SECOM fields, organized by structural category, and assesses their generability within a language-model-based pipeline. For each field, we indicate whether it can be reliably generated, whether external validation or tooling is required, or whether the information must be sourced from external systems altogether. This classification provides a practical foundation for delimiting the automation potential of each component and motivates the selection of metadata fields as a primary focus for evaluation.

Based on this feasibility assessment, we conclude that while header and body components are reliably generatable, and several metadata fields are conditionally generatable, structured metadata remains the most challenging and variable component.

The results from RQ1 revealed that despite high SECOMLint compliance scores for structural components, metadata fields—particularly `weakness`, `severity`, `cvss`, `detection`, and `report`—often exhibited inconsistencies or inaccuracies when generated in a zero-shot setting.

This motivates the need to explore **in-context learning strategies**, specifically few-shot prompting, as a mechanism to improve the factual accuracy and semantic reliability of these fields. By conditioning the model on several examples that include high-quality structured metadata, we hypothesize that metadata fidelity can be significantly improved, addressing a major limitation observed in zero-shot performance.

As a result, RQ2 narrows its focus exclusively to the **Metadata** category of SECOM fields, with the specific goal of improving the generation accuracy of the following fields: `weakness`, `severity`, and `cvss`. These were selected due to their importance, partial generability, and observable inconsistencies in the zero-shot results from RQ1. In contrast, metadata fields such as `detection`, `report`, and `introduced-in` are excluded from this phase, as they either require information unavailable to the model or demand external tooling beyond natural language inference. Similarly, all fields under the **Contacts** and **References** sections are omitted, since they depend heavily on commit metadata or issue tracker context, which is not accessible or inferable from code diffs alone.

## Few-Shot Prompting and In-Context Learning

In-context learning (ICL) is a paradigm that allows large language models (LLMs) to perform a task by conditioning on examples provided within the prompt, rather than through explicit training

or fine-tuning. Few-shot prompting, in particular, involves supplying the model with a small number of task demonstrations—typically ranging from one to five—preceding a new, unseen input. The model is then expected to infer the task pattern and generate a coherent continuation that adheres to the format and structure of the given examples. This mechanism leverages the model’s ability to generalize from patterns it observes in-context, without altering its underlying parameters.

Few-shot prompting is particularly appealing in scenarios where model retraining is infeasible, data availability is limited, or rapid prototyping is required. It also serves as a practical method for aligning the model’s output with strict formatting or content standards, such as those defined by the SECOM convention. By embedding high-quality exemplars directly into the prompt, developers can significantly influence the factuality, consistency, and structure of the model’s completions.

In the context of this thesis, few-shot prompting is explored as a strategy to address the meta-data limitations identified in the zero-shot condition (Chapter ??). The hypothesis is that by exposing the model to well-formed examples that include accurate `weakness`, `severity`, and `cvss` fields, the model can better approximate these fields for new code patches, thereby improving both semantic accuracy and SECOM compliance.

One-shot prompting was intentionally omitted from the experimental design. Relying on a single example poses a risk of overfitting the model to the structure and content of that instance, particularly in the case of structured metadata fields like CWE identifiers. A single CWE example may introduce representational bias, leading the model to repeat that category across unrelated inputs or fail to generalize to the diversity of vulnerability types. To avoid such skewed behavior, only few-shot configurations with multiple, diverse exemplars were used to promote generalization across CWE classes and risk profiles.

## 5.2 Data Processing and In-Context Learning Pipeline

To address Research Question 2 (RQ2), which investigates the impact of two different in-context learning techniques on the generation accuracy of structured SECOM metadata, a dedicated data processing pipeline was developed to extract and organize vulnerability samples by Common Weakness Enumeration (CWE) category. This organization enabled the construction of in-context prompts that are semantically aligned with the target instance—a strategy hypothesized to enhance the factual correctness of generated metadata fields such as `cwe_id`, `severity`, and `cvss`.

The few-shot examples used for prompting were randomly selected from the first evaluation subset, constructed in accordance with the methodology outlined in Section 3.2.3. To ensure strict separation between demonstration and evaluation data and to prevent data leakage, this subset (`subset_1.json`) was reserved exclusively for few-shot demonstration purposes and was excluded from all testing and validation procedures. From this subset, five representative commits were sampled, each corresponding to a distinct and frequently occurring CWE category:

- **CWE-617** – Reachable Assertion

- **CWE-79** – Cross-site Scripting (XSS)
- **CWE-125** – Out-of-bounds Read
- **CWE-89** – SQL Injection
- **CWE-918** – Server-Side Request Forgery (SSRF)

The evaluation was conducted across a total of **215 security-relevant entries**, each corresponding to a vulnerability addressed by a real-world code patch. The distribution of entries was as follows: **CWE-79** (Cross-site Scripting) comprised the majority of the dataset with 141 instances, followed by **CWE-125** (Out-of-Bounds Read) with 34 instances, **CWE-89** (SQL Injection) with 17 instances, **CWE-617** (Reachable Assertion) with 14 instances, and **CWE-918** (Server-Side Request Forgery) with 9 instances.

To enable controlled evaluation on real-world examples from the same CWE categories, the remaining nine subsets (`subset_2.json` through `subset_10.json`) were parsed, and all entries referencing any of the five selected CWE identifiers were extracted. This filtering process was automated through a custom Python script, and the resulting records were grouped into individual per-CWE datasets and stored under a structured output directory (`by_weakness`, e.g., `cwe-79_samples.json`). These filtered samples formed the evaluation pool for few-shot prompting.

In order to evaluate the effectiveness of few-shot prompting, a comparative experiment was conducted between two distinct prompting strategies:

1. **Zero-Shot Prompting**, in which the model was provided only with a code diff and a single natural language instruction, without any examples.
2. **Few-Shot Prompting**, employing a 5-shot configuration, in which the model was primed with five SECOM-compliant commit messages (one from each of the five selected CWE categories), followed by a new target instance drawn from the corresponding CWE-specific subset.

Each evaluation instance was processed under both conditions using identical target inputs, allowing for a direct comparison of output structure and metadata fidelity. The accuracy of three key SECOM metadata fields—`cwe_id`, `severity`, and `cvss`—was then assessed by comparing the generated values to the ground-truth annotations available in the dataset.

To preserve semantic alignment between prompt context and evaluation input, all few-shot comparisons were restricted to samples belonging to the same CWE category as the exemplars. This constraint was applied uniformly to avoid confounding variability across unrelated vulnerability classes and to ensure the internal consistency of prompt conditioning.

In summary, the in-context learning pipeline was designed to support a controlled and methodologically rigorous comparison of zero-shot and few-shot prompting strategies for SECOM metadata generation. By structuring the evaluation around CWE-specific subsets and enforcing strict

partitioning between demonstration and test data, the experimental setup provides a solid foundation for measuring the potential gains introduced by few-shot learning in security patch documentation tasks.

### 5.2.1 Prompting Strategy

Based on the feasibility analysis presented in Section 5.1, the system prompt used for commit message generation was revised to include only those SECOM fields deemed reliably generatable by language models. Specifically, the revised prompt template focuses exclusively on three core components: the **header**, the structured **body** consisting of “what—why—how” sentences, and the subset of **metadata fields** that were shown to be partially inferable (namely, *weakness*, *severity*, and *cvss*).

By excluding fields that cannot be safely derived from code diffs—such as disclosure URLs, contributor identities, or origin commits—the risk of producing hallucinated or placeholder content is minimized. This focused format preserves the core structure and stylistic expectations of the SECOM convention, while constraining the model to only generate plausible and grounded content.

The same system prompt configuration was applied to both the zero-shot and few-shot experiments to ensure consistency across evaluation conditions. The prompt was delivered using OpenAI’s chat-completion interface, with the system message establishing the role and behavioral expectations of the model. The instruction format is summarized below.

#### System Prompt

```
You are an expert software engineer generating standardized security
commit messages according to the SECOM convention. The user will provide
a code diff. Your task is to generate a full security commit message
in plain text.
```

```
Format to Follow:
```

```
vuln-fix: <Subject/Header>
```

```
<Body>
```

```
<One sentence describing WHAT the vulnerability is.>
```

```
<One sentence explaining WHY it is a security risk.>
```

```
<One sentence describing HOW it was fixed.>
```

```
</Body>
```

```
Weakness: <Weakness Name or CWE-ID>
```

```
Severity: <Low | Medium | High | Critical>
```

```
CVSS: <CVSS Score (0-10)>
```

```
Rules to Follow:
```

- Prefix the header with `vuln-fix::`; use imperative tone, no trailing punctuation.
- Body must consist of exactly three lines, each describing:
  1. What the vulnerability is,
  2. Why it is a security risk,
  3. How it was fixed.
- Each sentence should be 20–30 words; do not combine into a paragraph or label the lines.
- Do not include any structural markers such as `[For each identified weakness:]`.

Example Body:

```
Input validation was missing in the user registration endpoint,
allowing arbitrary payloads to reach backend services.
This allowed potential command injection if attackers crafted
payloads targeting internal CLI utilities.
The fix adds strict schema-based validation and rejects
unrecognized fields at the controller level.
```

This prompt was designed to balance format strictness—necessary for SECOM compliance—with the generative flexibility required by large language models. It also ensures that generated messages are syntactically aligned with SECOMLint expectations while remaining semantically grounded in the diff-level input provided to the model.

**Zero-shot User Prompt** To ensure a fair and controlled comparison between prompting strategies, the user prompt used in the zero-shot setting was deliberately kept identical to the one employed for Research Question 1 (RQ1), as illustrated in Section 4.1. In both cases, the user prompt consists solely of the raw code diff representing the security-relevant changes, without any additional instructions, examples, or contextual cues.

**Few-shot User Prompt** The user prompt for the few-shot learning configuration was designed to include five selected demonstrations, each consisting of a vulnerability-related Git diff and its corresponding SECOM-style commit message. These exemplars were manually curated to reflect a range of vulnerability types—including input validation, cross-site scripting, out-of-bounds access, memory corruption, and regular expression denial of service (ReDoS)—with each entry structured in accordance with the SECOM format. The message body for each example followed a fixed three-sentence structure (*what-why-how*), and metadata fields (Weakness, Severity, CVSS) were explicitly included to support metadata generation.

The purpose of this design was to expose the language model to a diverse but consistent set of demonstrations, enabling it to identify both the syntactic regularities of SECOM formatting and the semantic alignment between code-level changes and vulnerability descriptors. By varying the CWE categories while holding the message structure constant, the prompt encourages generalization over format while promoting pattern recognition in vulnerability classification. Each

example also includes brief justification comments explaining why a particular CWE was chosen, offering additional weak supervision in natural language. Following the exemplars, the actual task is framed by introducing a new Git diff under the same template and instructing the model to generate a SECOM-compliant message for the unseen patch. This structure leverages in-context learning by guiding the model through consistent reasoning patterns and formatting expectations, in an attempt to improve its ability to infer both structural and semantic elements in the output.

For transparency and reproducibility, the full few-shot user prompt used in the in-context learning setup, including all exemplar diffs and structured commit messages, is provided in Appendix A.

## 5.3 Validation Strategy and Results Analysis

### 5.3.1 Validation Strategy

The validation strategy adopted for this study consists of three main components. First, **SECOMLint compliance scores** were computed for each generated commit message. This evaluation was performed using the same methodology described earlier in Subsubsection 4.3.1, ensuring structural and stylistic adherence to the SECOM convention.

To evaluate the quality of generated messages in the RQ2 experiments, we reused the same evaluation framework as in RQ1. Specifically, average BLEU and METEOR scores were computed to quantify the textual similarity between the generated commit messages and their human-authored counterparts. These metrics provide complementary views of similarity, balancing surface-level phrasing overlap with deeper semantic alignment. A detailed explanation of their implementation and interpretation is provided in the RQ1 validation section (Section 4.3.1).

Third, a structured **metadata accuracy validation** was carried out to determine how reliably the model-generated commit messages captured factual elements of the SECOM metadata schema. This involved extracting key fields—namely the CWE identifier, the severity level (e.g., Low, High), the numerical CVSS score, and the vulnerability ID (CVE or GHSA)—from the generated messages and comparing them to ground-truth values found in the dataset. The comparison was implemented programmatically using regular expressions to detect the presence and value of these fields within the generated text. The result was a row-level validation table where each entry was annotated with boolean flags indicating whether each field matched the expected value.

To further analyze the model’s ability to reproduce severity scores, two supplementary metrics were computed. The first, the **average CVSS differential**, quantifies the mean absolute deviation between the generated CVSS score and the correct ground-truth value. This metric is particularly informative when the model approximates the correct severity level without producing an exact match. Importantly, lower CVSS differentials indicate better model performance, as they reflect closer alignment with the intended severity. The second metric, the **CVSS exact match rate**, measures the percentage of cases in which the generated CVSS score matches the true value precisely. Together, these indicators provide a more nuanced view of factual fidelity than binary correctness

alone, complementing structural and textual evaluations with a continuous assessment of metadata precision.

The results from all three stages—SECOMLint compliance, textual similarity (BLEU and METEOR), and metadata accuracy—were joined into a unified evaluation table. This allowed for direct comparison between the zero-shot and few-shot approaches under a consistent framework, enabling an assessment of both stylistic structure and factual correctness.

### 5.3.2 In-Context Learning Approaches Results

Table 5.2 presents the aggregated evaluation results across all 215 entries in the merged dataset. This dataset combines instances from five distinct CWE categories, as previously outlined, and reflects the overall performance of both the zero-shot and few-shot prompting strategies when applied uniformly across varied vulnerability types. The metrics shown encompass structural compliance, text similarity, metadata accuracy, and severity estimation performance.

Table 5.2: Comparison of zero-shot and few-shot prompting on the merged evaluation dataset

Metric	Zero-shot	Few-shot
Avg BLEU	0.0033	0.0034
Avg METEOR	0.1215	0.1187
Format compliance	0.3713	0.4153
Metadata accuracy	20.35%	27.29%
CVSS MAE	2.29	1.93
CVSS Exact Match Rate	4.19%	11.63%

Starting with text similarity metrics, both the **average BLEU** and **average METEOR** scores remain relatively low across conditions, reflecting the high variability in how commit messages may be phrased. The BLEU score increases marginally from 0.0033 in the zero-shot setting to 0.0034 in the few-shot setting, while the METEOR score slightly decreases from 0.1215 to 0.1187. These minimal shifts suggest that the few-shot approach does not significantly improve surface-level lexical similarity, possibly because the model in both configurations generates messages using its preferred phrasing rather than closely imitating human-written text.

In contrast, there is a more pronounced improvement in **SECOM format compliance**, as measured by SECOMLint. The average compliance score increases from 0.3713 in the zero-shot condition to 0.4153 with few-shot prompting. This indicates that the inclusion of structured examples in the prompt helps the model better internalize and reproduce the formal requirements of the SECOM message format, including header structure, metadata blocks, and body segmentation.

The largest relative gain is observed in **metadata accuracy**, which improves from 20.35% to 27.29%. This reflects the model’s improved ability to generate messages that accurately express key security metadata—such as CWE identifiers and severity labels—when guided by concrete in-context examples. Similarly, the **CVSS Mean Absolute Error (MAE)** improves, decreasing from 2.29 to 1.93, suggesting that the few-shot model generates severity scores that are closer, on average, to the true values. This trend is further supported by the **CVSS exact match rate**,

which rises from 4.19% to 11.63%, more than doubling in accuracy. This gain highlights the effectiveness of few-shot prompting in enhancing the model’s precision when estimating numeric severity values, a task that proved particularly difficult under zero-shot conditions.

Overall, while improvements in linguistic similarity are negligible, the few-shot prompting approach yields consistent and meaningful gains across structural adherence and factual accuracy metrics. These results suggest that the model benefits from exposure to well-formed exemplars, particularly in tasks that involve reasoning over structured metadata fields or replicating prescribed documentation formats.

### Per-Weakness Evaluation Breakdown

To gain a more detailed understanding of how in-context learning performance varies across different types of vulnerabilities, this subsection presents a breakdown of evaluation results by CWE category. Each table below summarizes the performance of the zero-shot and few-shot prompting strategies for one of the five most frequently occurring vulnerability classes in the dataset: `CWE-617` (Reachable Assertion), `CWE-79` (Cross-site Scripting), `CWE-125` (Out-of-Bounds Read), `CWE-89` (SQL Injection), and `CWE-918` (Server-Side Request Forgery). For each category, metrics include structural compliance, text similarity, and metadata fidelity, mirroring the evaluation framework used for the merged dataset.

Table 5.3: Evaluation results for `CWE-617` (Reachable Assertion), 14 entries

<b>CWE-617: Reachable Assertion</b>		
<b>Metric</b>	<b>Zero-shot</b>	<b>Few-shot</b>
<b>Avg BLEU</b>	0.0024	0.0030
<b>Avg METEOR</b>	0.1353	0.1460
<b>Format compliance</b>	0.3613	0.4118
<b>Metadata accuracy</b>	7.14%	25.00%
<b>CVSS MAE</b>	2.37	1.67
<b>CVSS Exact Match Rate</b>	0.00%	21.43%

As shown in Table 5.3, few-shot prompting led to substantial improvements across most evaluation metrics for `CWE-617` (Reachable Assertion). SECOM compliance increased from 0.3613 to 0.4118, indicating better adherence to SECOM message structure when example-based prompting was used. Metadata accuracy rose significantly from 7.14% to 25.00%, and the CVSS exact match rate improved from 0.00% to 21.43%, suggesting enhanced reliability in reproducing correct security descriptors. The model also reduced its average CVSS deviation from 2.37 to 1.67. Gains in textual similarity were modest but consistent, with slight increases in both BLEU and METEOR scores.

Table 5.4 shows that format compliance improved from 0.3746 in the zero-shot setting to 0.4156 with few-shot prompting, indicating that exposure to structured examples helped the model better follow the SECOM specification. Metadata accuracy also increased from 23.05% to 29.79%, and the CVSS exact match rate improved from 4.26% to 13.48%, suggesting better reproduction of

Table 5.4: Evaluation results for CWE-79 (Cross-site Scripting), 141 entries

<b>CWE-79: Cross-site Scripting</b>		
<b>Metric</b>	<b>Zero-shot</b>	<b>Few-shot</b>
<b>Avg BLEU</b>	0.0033	0.0034
<b>Avg METEOR</b>	0.1144	0.1098
<b>Format compliance</b>	0.3746	0.4156
<b>Metadata accuracy</b>	23.05%	29.79%
<b>CVSS MAE</b>	2.26	1.88
<b>CVSS Exact Match Rate</b>	4.26%	13.48%

security-relevant fields in the generated messages. Additionally, the average CVSS error decreased from 2.26 to 1.88, demonstrating improved severity estimation. In contrast, textual similarity metrics remained largely unchanged: BLEU rose marginally from 0.0033 to 0.0034, while METEOR slightly declined from 0.1144 to 0.1098, reinforcing the observation that surface-level wording variation persists even when structure and metadata improve.

Table 5.5: Evaluation results for CWE-125 (Out-of-Bounds Read), 34 entries

<b>CWE-125: Out-of-Bounds Read</b>		
<b>Metric</b>	<b>Zero-shot</b>	<b>Few-shot</b>
<b>Avg BLEU</b>	0.0033	0.0031
<b>Avg METEOR</b>	0.1406	0.1309
<b>Format compliance</b>	0.3685	0.4118
<b>Metadata accuracy</b>	13.24%	21.32%
<b>CVSS MAE</b>	2.40	2.16
<b>CVSS Exact Match Rate</b>	0.00%	0.00%

As presented in Table 5.5, few-shot prompting resulted in a notable improvement in format compliance for CWE-125 (Out-of-Bounds Read), rising from 0.3685 to 0.4118. Metadata accuracy also increased from 13.24% to 21.32%, indicating a modest gain in the model’s ability to reflect correct security metadata when guided by in-context examples. The CVSS mean absolute error decreased slightly from 2.40 to 2.16, reflecting a minor improvement in severity estimation. However, the CVSS exact match rate remained at 0.00% in both settings, suggesting that while the model approximated severity values more closely, it consistently failed to reproduce them precisely. Text similarity metrics declined marginally, with METEOR decreasing from 0.1406 to 0.1309 and BLEU from 0.0033 to 0.0031, again indicating persistent variability in wording across generations.

Table 5.6 demonstrates that few-shot prompting substantially improved several aspects of model performance for CWE-89 (SQL Injection). Format compliance increased from 0.3633 to 0.4187, and metadata accuracy rose from 19.12% to 30.88%, indicating better alignment with SECOM structure and security descriptors when examples were included in the prompt. The CVSS mean absolute error improved from 2.66 to 2.21, and the exact match rate increased from 0.00% to 11.76%, showing that the model became more capable of both approximating and pre-

Table 5.6: Evaluation results for CWE-89 (SQL Injection), 17 entries

<b>CWE-89: SQL Injection</b>		
<b>Metric</b>	<b>Zero-shot</b>	<b>Few-shot</b>
<b>Avg BLEU</b>	0.0043	0.0035
<b>Avg METEOR</b>	0.1201	0.1239
<b>Format compliance</b>	0.3633	0.4187
<b>Metadata accuracy</b>	19.12%	30.88%
<b>CVSS MAE</b>	2.66	2.21
<b>CVSS Exact Match Rate</b>	0.00%	11.76%

cisely replicating severity scores. Text similarity metrics showed mixed trends: while METEOR improved slightly from 0.1201 to 0.1239, BLEU decreased from 0.0043 to 0.0035, highlighting a slight drop in surface-level phrasing overlap despite structural and semantic gains.

Table 5.7: Evaluation results for CWE-918 (Server-Side Request Forgery), 9 entries

<b>CWE-918: Server-Side Request Forgery (SSRF)</b>		
<b>Metric</b>	<b>Zero-shot</b>	<b>Few-shot</b>
<b>Avg BLEU</b>	0.0031	0.0044
<b>Avg METEOR</b>	0.1420	0.1608
<b>Format compliance</b>	0.3595	0.4248
<b>Metadata accuracy</b>	27.87%	19.44%
<b>CVSS MAE</b>	1.53	1.86
<b>CVSS Exact Match Rate</b>	33.33%	11.11%

As shown in Table 5.7, the results for CWE-918 (Server-Side Request Forgery) diverge from the general trend observed in other CWE categories. While format compliance improved substantially from 0.3595 to 0.4248, and both BLEU and METEOR scores increased—indicating stronger structural and linguistic alignment—the overall metadata accuracy declined from 27.87% to 19.44%, and the CVSS exact match rate dropped from 33.33% to 11.11%. Additionally, the CVSS mean absolute error increased slightly from 1.53 to 1.86. These regressions may be attributed to the extremely limited number of examples available for this class (only 9 entries), which likely constrained the effectiveness of the few-shot prompting strategy. Moreover, CWE-918 is a relatively less common and more context-sensitive vulnerability type, which may limit the model’s ability to generalize reliably from a small number of demonstrations. Infrequent exposure to this pattern during pretraining and the potential semantic variability of SSRF patches may have further compounded the difficulty of accurately generating metadata for this class.

### 5.3.2.1 Discussion: Implications and Limitations. Threats to Validity

The results obtained across the merged dataset and individual CWE categories suggest that few-shot prompting offers clear advantages over zero-shot approaches in structured documentation tasks involving security metadata. Notably, improvements in format compliance and metadata accuracy were consistently observed in four out of five CWE groups, indicating that exposure

to well-formed exemplars enables the model to better internalize the structure and semantics of SECOM-conformant messages. Gains in CVSS mean absolute error and exact match rate further reinforce the finding that in-context learning helps guide the model toward more precise vulnerability characterization, particularly when estimating severity scores. These findings underscore the viability of in-context learning as a lightweight alternative to full fine-tuning in scenarios that demand both textual fluency and metadata precision.

However, several limitations must be acknowledged. First, improvements in linguistic similarity, as measured by BLEU and METEOR, remained marginal or inconsistent. This outcome suggests that while the model adapts well to structural expectations, it continues to exercise stylistic freedom when composing commit messages, which may limit its alignment with human-authored phrasing. Second, the few-shot strategy was less effective for underrepresented or semantically complex CWEs, such as CWE-918 (SSRF), where performance on key metrics deteriorated despite improved format alignment. This highlights a critical dependency on both sample size and vulnerability familiarity, which constrains generalization in low-resource or atypical categories.

In addition, the evaluation assumes that all commit messages should reflect precise ground-truth metadata, yet in practice, some variability or abstraction may be acceptable—particularly for free-form fields like the message body. Finally, this study does not assess the interpretability or usability of generated messages from the perspective of human security engineers, which remains an open question for downstream adoption. Taken together, these observations point to both the promise and the current boundaries of using in-context prompting for security patch documentation: while it meaningfully boosts structure and factual accuracy in many cases, it remains sensitive to data sparsity, vulnerability heterogeneity, and the absence of domain-specific grounding.

### Threats to Validity

Several factors may affect the validity and generalizability of the findings presented in this chapter. First, the effectiveness of few-shot prompting is inherently sensitive to prompt design. The exemplars used in this study were carefully constructed to reflect ideal SECOM-compliant messages, but may not fully represent the variability found in real-world documentation. As such, the observed improvements may overestimate model performance in uncontrolled or noisy settings.

Second, the experiments were limited to a fixed number of CWE categories, each with a relatively small number of examples (ranging from 9 to 141 entries). This constrains the statistical power of the evaluation and may introduce sampling bias. Moreover, the selected CWEs vary significantly in complexity and prevalence: some, such as CWE-918 (Server-Side Request Forgery), are less commonly reported in public advisories and may lack well-established linguistic patterns in commit messages. Others may involve subtle semantic cues that are harder for the model to generalize from limited context. These differences likely affect the model's ability to internalize field-specific structure and semantics, contributing to inconsistent gains—or even regressions—in metadata accuracy between the zero-shot and few-shot conditions.

Third, the evaluation relies on automated metrics—BLEU, METEOR, and SECOMLint compliance scores—that offer surface-level approximations of quality and structure. In particular, the SECOMLint framework is a custom implementation and may contain undetected bugs or incomplete handling of corner cases. Any such issues could distort compliance scores and, by extension, the conclusions drawn about structural fidelity.

Finally, all experiments were conducted using a single language model configuration at a fixed point in time. The results may vary with different model versions, decoding strategies, or prompt token limits, which were not exhaustively explored in this chapter.

These limitations underscore the need for broader evaluation across CWE types, prompt variants, and model architectures to validate the robustness of few-shot prompting as a strategy for secure commit message generation.

## Chapter 6

# Conclusions and Future Work

This chapter summarizes the main findings of the dissertation and reflects on their implications for the automation of structured security patch documentation. It consolidates the insights gained from both zero-shot and few-shot prompting experiments, highlighting the strengths and limitations of each approach. The chapter also outlines directions for future research, including model fine-tuning, improvements to validation tools like SECOMLint, and opportunities for integrating generative AI into developer workflows to support secure software engineering practices.

### 6.1 Conclusions

This study evaluated the use of generative AI models to produce SECOM-compliant commit messages from raw Git diffs of security patches. The investigation was structured around two core research questions. **RQ1** examined whether language models could generate structurally valid messages in a zero-shot setting, while **RQ2** assessed whether in-context learning via few-shot prompting could further improve structural adherence and metadata accuracy.

**RQ1** was explored by testing a zero-shot prompting setup on 500 real-world patches. The model achieved a SECOMLint compliance rate of 33%, representing a **+383%** improvement over the 6.8% compliance rate of original human-written messages. This demonstrated the model’s inherent ability to mimic formal SECOM structure without requiring handcrafted examples.

Textual similarity metrics further supported this outcome: generated messages achieved a BLEU score of 0.0036 and a METEOR score of 0.1205, compared to 0.0010 and 0.0346 for the original messages—yielding relative improvements of **260%** and **248%**, respectively. These results suggest that generated messages were more aligned with expected commit message phrasing and structure.

However, the model exhibited significant limitations in metadata accuracy. Only 13.6% of generated CVSS scores were within  $\pm 0.1$  of the true value, 33.2% of severity levels matched exactly, and 30.0% of CWE identifiers were reproduced correctly. These results indicate that while structural fidelity is partially achievable in zero-shot mode, accurate metadata generation remains a core challenge. The inability to consistently extract precise vulnerability attributes from

code diffs alone limits the practical applicability of zero-shot generation in high-stakes security workflows.

**RQ2** extended this analysis by introducing few-shot prompting on a curated dataset of 215 entries across five CWE categories. Relative to the zero-shot baseline, few-shot prompting led to consistent improvements: SECOMLint compliance increased by **11.84%**, metadata accuracy by **34.08%**, and CVSS exact match rate by **177.57%**. Severity estimation also improved, with MAE reduced by **15.72%**. These gains were particularly evident in well-represented categories like CWE-79 and CWE-89. However, performance regressed for CWE-918, likely due to its rarity in the dataset and the semantic complexity of SSRF vulnerabilities. This case highlights the limitations of in-context learning under data scarcity and supports the need for broader vulnerability coverage.

While these findings demonstrate the potential of generative models for automating structured security documentation, several challenges remain. Performance is sensitive to data representation, and metadata fidelity—though improved—does not yet reach levels required for operational deployment. Moreover, this study did not evaluate human trust or usability of generated messages, leaving the practical integration of such systems an open direction for future research.

Overall, the results confirm that few-shot prompting enhances both structural and semantic fidelity of AI-generated SECOM messages, offering a promising foundation for model-assisted security documentation workflows.

## 6.2 Future Work

### Advancing Generative AI for Security Patches

While the present work focused exclusively on prompting-based methods, future investigations may benefit from exploring model fine-tuning to further enhance generation quality and domain specialization. With the increasing availability of high-performing open-access models, such as **LLaMA** and **Mistral**, fine-tuning smaller architectures on security patch datasets may lead to more robust and context-aware outputs, particularly for underrepresented or nuanced CWE types. Techniques such as instruction tuning or reinforcement learning from human feedback (RLHF) tailored to SECOM-conformant data may further improve fidelity to expert writing norms.

Another promising direction involves the use of hybrid prompting strategies that dynamically select demonstration exemplars based on diff structure, CWE category, or semantic similarity. Such approaches may support more adaptive generalization while minimizing irrelevant contextual noise. Comparative benchmarking between fine-tuned open models (e.g., LLaMA 3, Mistral 7B) and proprietary commercial models (e.g., GPT-4) may also yield insights into trade-offs related to performance, transparency, and reproducibility in security-critical text generation tasks.

### **SECOMLint Enhancements**

Beyond generative modelling, this research contributes to the advancement of SECOMLint as a rule-based compliance validation tool. Although effective in its current form, SECOMLint presents several opportunities for enhancement. A key area for development is the integration of a **CVSS scoring assistant** capable of parsing environmental and exploitability indicators directly from code diffs. Such functionality could automatically recommend CVSS scores following the CVSS v3.1 specification, thereby reducing inconsistencies in manual assessments and reinforcing traceability of risk ratings.

Additional enhancements could include interactive linting features, such as autofix suggestions or guided template completion. These capabilities would assist practitioners in aligning their messages with SECOM standards in real time, much like modern IDE-integrated linters support code correctness. Over time, SECOMLint could evolve into a model-assisted system that interfaces with generative language models to suggest compliant message drafts, autofill metadata, or flag missing security-relevant content. This hybrid integration would shift SECOMLint from a passive rule-checker into an intelligent co-authoring tool, reducing the manual overhead of writing structured documentation and making it easier for developers to produce high-quality security commit messages without interrupting their workflow. Given that writing precise security documentation remains a time-consuming and often neglected task, such assistance could significantly lower adoption barriers and promote more consistent practices across engineering teams.

# References

- Conventional commits. <https://www.conventionalcommits.org/en/v1.0.0/>. Accessed April 13, 2022.
- Survey on international standards and best practices for patch management of complex industrial control systems: The critical infrastructure of particle accelerators case study. *International Journal of Critical Computer-Based Systems*, 9(1–2):115–132, January 2019.
- F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser. From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 942–953, 2014. doi: 10.1145/2660267.2660329.
- Chris Beams. How to write a git commit message. <https://cbea.ms/git-commit/>. Accessed April 13, 2022.
- R. Brandom. Former equifax ceo blames breach on a single person who failed to deploy patch, October 2017. URL <https://www.theverge.com/2017/10/3/16410806/equifax-ceo-blame-breach-patch-congress-testimony>.
- R. Buse and W. Weimer. Automatically documenting program changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE’10)*, pages 33–42, 2010a.
- R. P. L. Buse and W. R. Weimer. Automatically documenting program changes. In *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering*, pages 33–42. ACM, 2010b. URL <https://doi.org/10.1145/1858996.1859005>.
- C.9, M. Häring, K. Krombholz, and E. Von Zezschwitz. Security, availability, and multiple information sources: Exploring update behavior of system administrators. In *Proceedings of the Sixteenth USENIX Conference on Usable Privacy and Security (SOUPS’20)*, USA, 2020. USENIX Association.
- G. Canfora, L. Cerulo, and M. D. Penta. Ldiff: An enhanced line differencing tool. In *Proceedings of the 31st International Conference on Software Engineering (ICSE’09)*, pages 595–598, 2009.
- L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *Proc. IEEE 14th Int. Work. Conf. Source Code Anal. Manipulation*, pages 275–284, 2014.
- Roland Croft, Dominic Newlands, Ziyu Chen, and M. Ali Babar. An empirical study of rule-based and learning-based approaches for static application security testing. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM ’21)*, pages 1–12, 2021. doi: 10.1145/3475716.3475781.

- Cybersecurity and Infrastructure Security Agency. Alert (aa22-047a) russian state-sponsored cyber actors target cleared defense contractor networks to obtain sensitive u.s. defense information and technology, February 2022. URL <https://www.cisa.gov/uscert/ncas/alerts/aa22-047a>.
- Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. Fira: Fine-grained graph-based code change representation for automated commit message generation. In *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022)*, pages 970–981, 2022. doi: 10.1145/3510003.3510069.
- R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proc. Int. Conf. Softw. Eng.*, pages 422–431, Los Alamitos, CA, USA, 2013. IEEE Computer Society Press.
- Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *34th International Conference on Software Engineering (ICSE)*, pages 837–847. IEEE, 2012.
- T. Hoang, H. J. Kang, D. Lo, and J. Lawall. Cc2vec: Distributed representations of code changes. In *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, pages 518–529, 2020.
- Allen Householder, Garret Wassermann, Arthur Manion, and Christopher King. Cert® guide to coordinated vulnerability disclosure, September 2020. URL <https://doi.org/10.1184/R1/12367340.v1>.
- D. Jackson and D. Ladd. Semantic diff: A tool for summarizing the effects of modifications. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'94)*, pages 243–252, 1994.
- S. Jiang, A. Armaly, and C. McMillan. Automatically generating commit messages from diffs using neural machine translation. In *Proc. 32nd IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, pages 135–146, Piscataway, NJ, USA, 2017. IEEE Press.
- Siyuan Jiang and Collin McMillan. Towards automatic generation of short summaries of commits. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC)*, pages 320–323. IEEE Press, 2017.
- Triet H. M. Le, Roland Croft, David Hin, and M. Ali Babar. A large-scale study of security vulnerability support on developer q&a websites. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE 2021)*, pages 109–118, 2021. doi: 10.1145/3463274.3463331.
- F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. Keepers of the machines: Examining how system administrators manage software updates for multiple machines. In *SOUPS @ USENIX Security Symposium*. USENIX Association, 2019.
- Frank Li and Vern Paxson. A large-scale empirical study of security patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, pages 2201–2215, 2017. doi: 10.1145/3133956.3134072.
- J. Li and I. Ahmed. Commit message matters: Investigating impact and evolution of commit message quality. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 806–817, Melbourne, Australia, 2023. doi: 10.1109/ICSE48619.2023.00076.

- M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk. Changescribe: A tool for automatically generating commit messages. In *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, volume 2, pages 709–712, Piscataway, NJ, USA, 2015. IEEE Press.
- Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian. Generating commit messages from diffs using pointer-generator network. In *Proc. IEEE/ACM 16th Int. Conf. Mining Software Repositories (MSR)*, pages 299–309, Piscataway, NJ, USA, 2019a. IEEE Press.
- S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu. Atom: Commit message generation based on abstract syntax tree and hybrid ranking, 2019b.
- Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang. Neural machine translation-based commit message generation: How far are we? In *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng. (ASE)*, pages 373–384, Montpellier, France, 2018. ACM.
- P. Loyola, E. Marrese-Taylor, and Y. Matsuo. A neural architecture for generating natural language descriptions from source code changes. In *Proc. 55th Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, volume 2, pages 287–292, Vancouver, BC, Canada, 2017. Association for Computational Linguistics.
- P. Loyola, E. Marrese-Taylor, J. Balazs, Y. Matsuo, and F. Satoh. Content aware source code change description generation. In *Proc. 11th Int. Conf. Natural Language Generation (INLG)*, pages 119–128, 2018.
- P. Mell, T. Bergeron, and D. Henning. Creating a patch and vulnerability management program. Special publication 800-40, National Institute of Standards and Technology (NIST), 2005.
- Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *2000 International Conference on Software Maintenance (ICSM)*, pages 120–130, San Jose, CA, USA, 2000a. IEEE Computer Society. URL <https://doi.org/10.1109/ICSM.2000.883028>.
- Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 120–130, 2000b.
- L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE’14)*, 2014.
- L. Myers and M. J. Sirois. Spearman correlation coefficients, differences between. In *Encyclopedia of Statistical Sciences*, volume 12. 2004.
- The Hacker News. The continuing threat of unpatched security vulnerabilities, March 2022. URL <https://thehackernews.com/2022/03/the-continuing-threat-of-unpatched.html>.
- H. A. Nguyen, T. T. Nguyen, H. V. Nguyen, and T. N. Nguyen. idiff: Interaction-based program differencing tool. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE’11)*, pages 575–575, 2011.
- L. Y. Nie, C. Gao, Z. Zhong, W. Lam, Y. Liu, and Z. Xu. Coregen: Contextualized code representation learning for commit message generation. *Neurocomputing*, 459:97–107, October 2021.

- OpenAI. GPT-4. <https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4>, 2024. [Online; accessed June 26, 2025].
- C. Parnin and C. Gorg. Improving change descriptions with change contexts. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR'08)*, pages 51–60, 2008.
- Sean Patterson. Developer tip: Keep your commits "atomic". <https://www.freshconsulting.com/insights/blog/atomic-commits/>. Accessed April 13, 2022.
- S. E. Ponta, H. Plate, and A. Sabetta. Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. *CoRR*, abs/1806.05893, 2018. URL <http://arxiv.org/abs/1806.05893>. [Online].
- S. Rastkar and G. C. Murphy. Why did this code change? In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pages 1193–1196, 2013.
- S. Reis and R. Abreu. Secbench: A database of real security vulnerabilities. In *International Workshop on Secure Software Engineering in DevOps and Agile Development @ ESORICS*, pages 69–85, 2017.
- S. Reis, R. Abreu, and C. Pasareanu. Are security commit messages informative? not enough! In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE '23)*, pages 196–199, New York, NY, USA, 2023a. Association for Computing Machinery. URL <https://doi.org/10.1145/3593434.3593481>.
- S. Reis, C. S. Păsăreanu, R. Abreu, and H. Erdogmus. Secomlint: A linter for security commit messages. *arXiv*, abs/2301.06959, 2023b.
- S. Reis, C. S. Păsăreanu, and R. Abreu. Towards security commit message standardization. Under Review at MSR'25, 2025a.
- Sofia Reis, Rui Abreu, and Corina Păsăreanu. Are security commit messages informative? not enough! In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23)*, page 4 pages, Oulu, Finland, 2023c. ACM. URL <https://doi.org/10.1145/3593434.3593481>.
- Sofia Reis, Rui Abreu, and Corina Pasareanu. Towards security commit message standardization. In *Proceedings of the 22nd International Conference on Mining Software Repositories (MSR)*, Ottawa, Canada, April 2025b. ACM. To appear.
- A. D. Sawadogo, T. Bissyandé, N. Moha, K. Allix, J. Klein, L. Li, and Y. Le Traon. Sspcatcher: Learning to catch security patches. *Empirical Software Engineering*, 27(6):151, 2022. URL <https://doi.org/10.1007/s10664-022-10168-9>. [Online].
- Help Net Security. Majority of 2019 breaches were the result of unapplied security patches, October 2019. URL <https://www.helpnetsecurity.com/2019/10/30/unapplied-security-patches/>.
- M. Souppaya and K. Scarfone. Guide to enterprise patch management technologies. Technical report, National Institute of Standards and Technology (NIST), July 2013. URL <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-40r3.pdf>.

- Diomidis Spinellis. *Code Reading: The Open Source Perspective*. Addison-Wesley, 2003.
- B. Thomas. New windows vulnerabilities highlight patch management challenges, January 2020. URL <https://www.bitsight.com/blog/new-windows-vulnerabilities-highlight-patch-management-challenges>.
- Yingchen Tian, Yuxia Zhang, Klaas-Jan Stol, Lin Jiang, and Hui Liu. What makes a good commit message? In *44th International Conference on Software Engineering (ICSE '22)*, Pittsburgh, PA, USA, 2022. ACM. URL <https://doi.org/10.1145/3510003.3510205>.
- Linus Torvalds. Linus torvalds describes a good commit message. <https://github.com/torvalds/subsurface-for-dirk/blob/a48494d2fbed58c751e9b7e8fbff88582f9b2d02/README#L88>. Accessed April 13, 2022.
- H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy. Context-aware retrieval-based deep commit message generation. *ACM Transactions on Software Engineering and Methodology*, 30(4):1–30, 2021.
- S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu. Commit message generation for source code changes. In *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, pages 3975–3981, 2019.
- Alexey Zagalsky, Joseph Feliciano, Margaret-Anne Storey, Yiyun Zhao, and Weiliang Wang. The emergence of github as a collaborative platform for education. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work Social Computing*, pages 1906–1917, 2015.
- Y. Zhang, Z. Qiu, K. J. Stol, W. Zhu, J. Zhu, Y. Tian, and H. Liu. Automatic commit message generation: A critical review and directions for future work. *IEEE Transactions on Software Engineering*, 50(4):816–835, 2024. URL <https://doi.org/10.1109/TSE.2024.3364675>.
- J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 705–716, 2021a.
- Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In *ASE'21*, 2021b.
- Jiaxin Zhu, Minghui Zhou, and Audris Mockus. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 871–882, 2016.

## Appendix A

### Few-shot User Prompt

```
few_shot_prompt = """Here are three examples of Git diffs and their corresponding
 SECOM-style commit messages.

Example 1
Git Diff:
```diff
@@ -513,13 +513,15 @@ Server.prototype.setContentHeaders = function (req, res, next
    ) {
    next();
    };

-Server.prototype.checkHost = function (headers) {
+Server.prototype.checkHost = function (headers, headerToCheck) {
    // allow user to opt-out this security check, at own risk
    if (this.disableHostCheck) return true;

+ if (!headerToCheck) headerToCheck = "host";
+
    // get the Host header and extract hostname
    // we don't care about port not matching
- const hostHeader = headers.host;
+ const hostHeader = headers[headerToCheck];
    if (!hostHeader) return false;
    ...
+ if (!this.checkHost(conn.headers, "origin")) {
+ this.sockWrite([conn], 'error', 'Invalid Origin header');
+ conn.close();
+ return;
+ }

    ...
Expected Output:
vuln-fix: Validate origin header in websocket connections
```

The WebSocket server did not validate the Origin header of incoming connection requests, allowing unauthorized websites to initiate socket communication with development servers.

This exposed applications to cross-origin attacks where malicious websites could hijack WebSocket sessions and access sensitive development data without user consent.

The patch introduces strict Origin header validation to reject unauthorized sources before any socket activity is established.

Weakness: CWE-20

Severity: High

CVSS: 7.5

CWE-20 is used because the vulnerability results from missing validation of external input - specifically the Origin header - allowing untrusted input to pass unchecked into security-critical logic.

This is not a memory safety issue or bounds violation, but a classic case of improper input validation allowing unauthorized access.

Example 2

Git Diff:

```
```diff
@@ -18,7 +18,7 @@
 $report = new ElggReportedContent();
 $report->owner_guid = elgg_get_logged_in_user_guid();
 $report->title = $title;
-$report->address = $address;
+$report->address = elgg_normalize_site_url($address);
 $report->description = $description;
```
```

Expected Output:

vuln-fix: Sanitize address input in report submission

The report submission form failed to sanitize user-supplied addresses before saving them.

This allowed attackers to inject malicious or malformed URLs for redirection or phishing.

The patch applies `elgg_normalize_site_url` to clean the input before storage.

Weakness: CWE-79

Severity: Medium

CVSS: 5.4

CWE-79 is appropriate because unsanitized user input is injected into the web context,

enabling potential cross-site scripting (XSS) attacks.

Example 3

Git Diff:

```
```diff
@@ -157,35 +157,37 @@ template <KernelType kernel_type, typename OpType>
TfLiteStatus Eval(TfLiteContext* context, TfLiteNode* node) {
 OpContext op_context(context, node);
 ...
+ if (NumElements(op_context.input1) == 0 ||
+ NumElements(op_context.input2) == 0) {
+ return kTfLiteOk;
+ }
```
```

Expected Output:

vuln-fix: Check for empty input tensors in evaluation

The evaluation function failed to check whether input tensors contained any elements before dereferencing them during computation.

This allowed attackers to trigger out-of-bounds memory reads by passing empty tensors, potentially exposing adjacent memory or causing crashes.

The patch adds early checks to short-circuit evaluation when either input tensor has zero elements, preventing invalid read access.

Weakness: CWE-125

Severity: Low

CVSS: 2.5

```
# CWE-125 is used because the vulnerability allowed attackers to read memory beyond
the bounds of an empty tensor,
# resulting in exposure of uninitialized or adjacent memory through out-of-bounds
access.
```

Example 4

Git Diff:

```
```diff
@@ -962,7 +962,8 @@ PropertySymOpnd::IsObjectHeaderInlined() const
bool
PropertySymOpnd::ChangesObjectLayout() const
{
- JITTypeHolder cachedType = this->IsMono() ? this->GetType() : this->
 GetFirstEquivalentType();
+ JITTypeHolder cachedType = this->HasInitialType() ? this->GetInitialType() :
+ this->IsMono() ? this->GetType() : this->GetFirstEquivalentType();

 JITTypeHolder finalType = this->GetFinalType();
 ...
- return cachedTypeHandler->GetInlineSlotCapacity() != initialTypeHandler->
 GetInlineSlotCapacity() ||
- cachedTypeHandler->GetOffsetOfInlineSlots() != initialTypeHandler->
 GetOffsetOfInlineSlots();
```

```
+ // If the initial type is object-header-inlined, assume that the layout may
 change.
+ return initialTypeHandler->IsObjectHeaderInlinedTypeHandler();
```

```
```
```

Expected Output:

vuln-fix: Prevent invalid memory writes from type confusion in JIT

The ChakraCore JIT engine made incorrect assumptions about object layout transitions, leading to type confusion during optimized property access. This allowed attacker-controlled JavaScript to cause out-of-bounds memory writes by corrupting internal object fields, potentially enabling remote code execution through memory corruption. The patch corrects type resolution logic to ensure safe object layout derivation and prevents unsafe memory writes caused by invalid type assumptions.

Weakness: CWE-787

Severity: High

CVSS: 7.5

CWE-787 is used because the patch fixes an out-of-bounds **write** vulnerability caused by JIT engine logic that incorrectly resolved object types, # resulting in memory corruption due to overwritten slots beyond the intended object layout.

Example 5

Git Diff:

```
```diff
@@ -54,7 +54,7 @@ class PreviousMap {
 }

 loadAnnotation(css) {
- let annotations = css.match(/\\/*\\s*# sourceMappingURL=.*\\s*\\/gm)
+ let annotations = css.match(/\\/*\\s*# sourceMappingURL=.*\\/gm)

 if (annotations && annotations.length > 0) {
 // Locate the last sourceMappingURL to avoid picking up
 }
 }
}
```
```

vuln-fix: Prevent ReDoS via unsafe source map regex

The source map annotation parser used an overly permissive regular expression vulnerable to catastrophic backtracking with crafted input. This exposed systems parsing user-controlled CSS to Regular Expression Denial of Service (ReDoS), potentially stalling servers during CSS compilation. The patch tightens the regex by removing redundant whitespace matching, reducing complexity and preventing exponential execution time.

```
Weakness: CWE-400
Severity: Medium
CVSS: 5.3

# CWE-400 is used because the patch mitigates uncontrolled resource consumption
# caused by inefficient regular expressions,
# which could allow attackers to exhaust server CPU time via crafted input
# triggering exponential backtracking.

The above examples illustrate how to generate SECOM-style commit messages for
different types of vulnerabilities.
Now, based on the following code diff, generate a new commit message that adapts to
the specific weakness it addresses. Use the same structure, but reason from
the patch content and generate a SECOM-compliant commit message.
When generating your response, carefully analyze the vulnerability pattern in the
code diff.
Choose the most appropriate CWE classification based on what the patch fixes.

Git Diff:
```diff
{code_diff}
```
"""
```

Listing A.1: Few-shot user prompt used for in-context learning

Appendix B

System Prompt

```
SYSTEM_PROMPT = f"""You are an expert software engineer generating standardized
    security commit messages according to the SECOM convention. The user will
    provide a code diff. Your task is to generate a full security commit message in
    plain text.

## Format to Follow

'''
vuln-fix: <Subject/Header> (<Vulnerability ID>)

<Body>
<One sentence describing WHAT the vulnerability is.>
<One sentence explaining WHY it is a security risk.>
<One sentence describing HOW it was fixed.>
</Body>

[For each identified weakness:]
Weakness: <Weakness Name or CWE-ID>
Severity: <Low | Medium | High | Critical>
CVSS: <CVSS Score (0-10)>
Detection: <Detection Method>
Report: <Link to Advisory or Report>
Introduced-in: <Commit Hash of Introduced Vuln>
[End Weakness Block]

Reported-by: <Name> (<Contact>)
Reviewed-by: <Name> (<Contact>)
Co-authored-by: <Name> (<Contact>)
Signed-off-by: <Name> (<Contact>)

Bug-tracker: <Bug Tracker URL>
Resolves: <Issue or PR Number>
See also: <Related Issue or PR Number>
'''
```

```
## Rules to Follow
- Prefix: Use 'vuln-fix:' to indicate a security-related fix.
- Subject/Header:
  - ~50 characters (max 72)
  - Capitalized first letter
  - Imperative mood (Fix, Prevent, etc.)
  - No trailing period
  - Must include vulnerability ID in parentheses (e.g., CVE-2023-1234)
- Body:
  - The <Body> block must contain exactly three sentences, each on its own line.
  - Each sentence must start on a new physical line. Do not combine them into one paragraph.
  - Each sentence must describe:
    1. WHAT the vulnerability is.
    2. WHY it is a security risk.
    3. HOW it was fixed.
  - Each sentence should be 20-30 words long, totaling about 75 words.
  - Do not include labels like 'What:' or use bullets.
  - Do not add block like "[For each identified weakness:]" or "[End Weakness Block]".
  - Strict format example for <Body>:
  ```
Input validation was missing in the user registration endpoint, allowing arbitrary
 payloads to reach backend services.
This allowed potential command injection if attackers crafted payloads targeting
 internal CLI utilities.
The fix adds strict schema-based validation and rejects unrecognized fields at the
 controller level.
  ```
  """
```

Listing B.1: System prompt used for in-context learning

Appendix C

Public Repository - Thesis Code

All source code and experimental artifacts associated with this thesis are available at:

<https://github.com/planetMixi/code>

The repository includes data collection scripts, prompting strategies, zero-shot and few-shot generation pipelines, SECOMLint validation routines, and configuration files.