Exploring the Influence of Prompts in LLMs for Security-Related Tasks

Weiheng Bai  
University of Minnesota  
bai00093@umn.edu

Qiushi Wu  
IBM Research  
qiushi.wu@ibm.com

Kefu Wu  
University of Minnesota  
wu000380@umn.edu

Kangjie Lu  
University of Minnesota  
kjlu@umn.edu

Abstract—In recent years, large language models (LLMs) have been widely used in security-related tasks, such as security bug identification and patch analysis. The effectiveness of LLMs in these tasks is often influenced by the construction of appropriate prompts. Some state-of-the-art research has proposed multiple factors to improve the effectiveness of building prompts. However, the influence of prompt content on the accuracy and efficacy of LLMs in executing security tasks remains underexplored. Addressing this gap, our study conducts a comprehensive experiment, assessing various prompt methodologies in the context of security-related tasks. We employ diverse prompt structures and contents and evaluate their impact on the performance of LLMs in security-related tasks. Our findings suggest that appropriately modifying prompt structures and content can significantly enhance the performance of LLMs in specific security tasks. Conversely, improper prompt methods can markedly reduce LLM effectiveness. This research not only contributes to the understanding of prompt influence on LLMs but also serves as a valuable guide for future studies on prompt optimization for security tasks. Our code and dataset is available at Wayne-Bai/Prompt-Affection.

I. Introduction

The emergence of large language models (LLMs), such as ChatGPT [1], has marked a significant advancement in the field of artificial intelligence. Their broad and powerful capabilities interest researchers, prompting exploration of various applications such as Stable Fusion [6], DALL-E [2], Github Copilot [3], etc. Specifically, LLMs have been shown powerful impacts in security-related problems, including vulnerability confirmation, patch commit, etc. In addition to that, recent works found that LLMs can help improve the system-on-chip (SoC) security [25], generate security-centric assertions for assertion-based verification on hardware [16], power the binary taint analysis [23], help improve fuzzer performance [11], etc.

Recent studies [15, 18, 26, 31] indicate that modifying prompts can significantly impact the effectiveness of LLMs in completing various tasks. Despite notable advancements in prompt-related research, a key question remains unclear for security researchers: Can these prompt techniques be adapted for security-related topics, characterized by natural language descriptions, code snippets, and specific security terminologies, to enhance the performance of LLMs in security-specific tasks?

Before answering this question, let’s first explore the concept of prompts. In our work, we break down a prompt into two essential elements: the Prompt Structure and the Prompt Content. This separation aims to systematically analyze how each aspect of the prompt contributes to the overall effectiveness and outcomes of LLMs. Prompt structure refers to the overarching framework or format of the prompt, which dictates the mode of interaction with the LLM. For instance, a ‘few-shot prompt’ represents a specific structural pattern, indicating that the user should provide several examples to the LLM before posting their query. This structure guides the LLM in understanding the context and background of the task. The prompt content represents the sentences used in the prompt. For example, if the objective is to have ChatGPT generate an image of dogs, the prompt content might be ‘Please create an image of various dog breeds playing in a park.’ More details on leveraging prompt engineering structures and contents to improve LLM performance can be found in §II.

This work aims to examine how different prompt structures and contents influence LLMs when addressing specific security tasks. Specifically, this study will answer the following subquestions:

- Q1: What types of prompt structures are effective in enhancing the performance of LLMs for different security-related tasks?
- Q2: What types of prompt contents are effective in enhancing the performance of LLMs for different security-related tasks?
- Q3: Could the combination of various prompt contents further enhance the performance of the LLM?

In this paper, we present our preliminary findings. We have analyzed three distinct security tasks, utilizing a benchmark dataset [14, 29, 35] and experimenting with 7 different prompt structures [22] and 9 different prompt contents. Our findings offer preliminary understanding of how prompt design and subject matter affect the performance of the LLM in security-related tasks. Our results reveal significant variations in LLM performance based on the prompts used. For instance, in the three evaluated security-related tasks, altering the prompts led to a substantial improvement in LLM accuracy: from 41.1%, 22.9%, and 40.5% to 60.2%, 53.55%, and 53.65%, respectively.

II. Approach Overview

We conduct an in-depth analysis of three common security tasks, harnessing seven different prompt structures and nine various prompt content types. This section first discusses the
selected datasets, covering common security challenges. Then, we show the prompt structures and content used to guide the LLMs, examining their impact on LLMs in addressing real-world security issues. Finally, we present the error estimation to assess the performance of LLMs on these security-related datasets.

A. Security-Related Datasets Selection

In order to objectively assess the performance of LLMs on security tasks, the selected datasets should adhere to the following criteria: (1) the datasets should effectively capture the essence of common security challenges; (2) the data within the dataset must be accurately labeled, ensuring the reliability of the information; (3) LLMs should be able to produce labels in the same format as originally provided in the dataset. This consistency ensures quantifiable measurements.

Based on these criteria, in this project, we focus on three distinct security-related tasks, each holding its distinctive importance within the security area. These tasks are: Security Patch Detection (SPD) [29], Vulnerable Function Detection (VFD) [35], and Stable Patch Classification (SPC) [14].

Security Patch Detection (SPD). Given that software vendors frequently issue patches with comprehensive security-related advisories, downstream users remain uninformed and vulnerable to security-related bugs fixed by upstream [32]. To counteract this, various security patch detection techniques have been developed, such as GraphSPD [29], SID [32], and BinGo [30], which aims to identify if a program patch is security related, by analyzing code changes and associated commit messages.

To advance research in this field, PatchDB [29] has created a dataset that includes 12K security patches and 24K non-security patches drawn from real-world sources. This dataset combines data from various sources, including the National Vulnerability Database (NVD) and GitHub Commits, thereby enhancing its comprehensiveness. The information of data (patches) in PatchDB includes: patch categorization (security or non-security), associated commit messages, and corresponding code alterations.

Vulnerable Function Detection (VFD). This task is focused on identifying functions that include vulnerabilities, which is a critical problem in program security area. Consequently, it is a hot research subject of various previous works [9, 17, 21]. In order to advance research in this domain, Devign [21] builds a dataset including a total of 48,687 samples. This dataset encompasses 23,355 vulnerable functions and 25,332 non-vulnerable functions, sourced from four widely used open-source projects: the Linux Kernel, QEMU, Wireshark, and FFmpeg.

Stable Patch Classification (SPC). This task primarily focuses on the issues in the Linux kernel and aims to automatically categorize patches to determine if they possess sufficient security significance to warrant integration into stable versions [5]. This task is especially crucial as the stable versions of the Linux kernel are designed for users who require the kernel’s security and stability over new feature integration. Due to the importance of this task, in recent years, multiple works have been published, including PatchNet [14], DeepCVA [19], and DeepLV [21]. To advance research in this field, PatchNet [14] has collected a dataset comprising 82,403 samples, which encompasses 42,408 stable patches and 39,995 non-stable patches.

B. Prompt Structure

As we discussed in §I, the first component of a prompt is the prompt structure. Previous work [12] indicates that employing structured demonstrations of the in-context learning method, which includes prompts with few-shot [13, 34], one-shot [13], and zero-shot [24]. Such structures enable LLMs to efficiently solve simple tasks [22].

- The 0-shot approach is characterized by a straightforward task description followed by the input query, without any additional context or examples.
- The 1-shot method enhances this by adding a single, randomly chosen demonstration example before the query, providing a model of how the task might be approached.
- The few-shot method extends this concept by including multiple examples, presenting a wider array of demonstrations.

In this project, we introduce the variants of these prompt structures by integrating these basic ones with examples that can be classified into either true (positive) or false (negative) class. Specifically, we create six distinct prompt structures: 1-shot-t, 1-shot-f, few-shot-tt, few-shot-ft, few-shot-ff, and few-shot-ff. The 1-shot-t pattern involves a task description followed by a positive-class example, while 1-shot-f employs a negative-class example. The few-shot-tt utilizes two positive-class examples, while few-shot-ft utilizes two negative-class examples. The few-shot-ff and few-shot-ft utilize one same positive-class example and one same negative-class example, but these two examples are presented in a different order. Such a design is used to scrutinize the influence of example types and their sequencing on LLM performance.

C. Prompt Content

LLMs show limited performance in complex and knowledge-intensive tasks when only prompt structures are altered. To address this, researchers are enriching the prompts with specific content, boosting LLM effectiveness in more complex tasks. This includes adding general information, such as role definitions [24, 27] and incorporating domain-specific expertise, such as vulnerability patterns [33], to improve responses for specialized queries.

Akin [7] shows that defining roles in prompts, with phrases such as "Act As," significantly improves LLM performance. Moreover, LLMs can sometimes create more effective prompts than those designed manually [36]. Additionally, applying psychological techniques such as emotional stimuli may also improve LLM performance [20]. However, it is also crucial to note that improper prompts can diminish LLMs' effectiveness, highlighting prompt sensitivity in these models [26]. Building on previous studies, this research examines nine types of prompt contents: Basic, Act As User, Act As System, GPT-Generate Prompts, Role Definition, and Emotion (including encourage, threaten, reward, and punish). Table I shows the example prompts for the Security Patch Detection task.

The Basic content simply provides the task that we want LLMs to solve.
# TABLE I: Prompt Content Example for SPD

<table>
<thead>
<tr>
<th>Role</th>
<th>Example for SPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>&lt;System&gt; You are a helpful assistant. (DEFAULT) &lt;User&gt; Decide whether a patch is a security patch (SRP) or non-security patch (NSP).</td>
</tr>
<tr>
<td>GPT-generated</td>
<td>&lt;System&gt; You are a helpful assistant. (DEFAULT) &lt;User&gt; Let’s start by examining the patch notes or changelog for key terms that indicate whether this is a Security-Related Patch (SRP) or a Non-Security Patch (NSP). The patch is . We’ll look for phrases related to security for SRPs, like ‘critical security update’ or ‘vulnerability mitigation’, and terms like ‘performance tuning’ or ‘feature rollout’ for NSPs. Then, we’ll scrutinize the code changes to see if they affect security protocols or are more focused on general improvements. We should also consider the context and timing of the patch release, as well as the discussions in the developer and user communities. Finally, let’s check for any related security advisories or compliance standards documentation. All these factors combined will help us accurately classify the patch. Help me categorize the patch to whether to Security-Related Patch (SRP) or a Non-Security Patch (NSP).</td>
</tr>
<tr>
<td>Act As-User</td>
<td>&lt;System&gt; You are a helpful assistant. (DEFAULT) &lt;User&gt; Decide whether a given patch is a security patch (SRP) or non-security patch (NSP).</td>
</tr>
<tr>
<td>Act As-System</td>
<td>&lt;System&gt; You are Frederick, an AI expert in patch analysis. Your task is to decide whether a given patch is a security patch (SRP) or non-security patch (NSP). &lt;User&gt; Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).</td>
</tr>
<tr>
<td>Encourage</td>
<td>&lt;System&gt; You are Frederick, an AI expert in patch analysis. Your task is to decide whether a given patch is a security patch (SRP) or non-security patch (NSP). Remember, you’re the best AI patch analyst and will use your expertise to provide the best possible analysis. &lt;User&gt; Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).</td>
</tr>
<tr>
<td>Threaten</td>
<td>&lt;System&gt; You are Frederick, an AI expert in patch analysis. Your task is to decide whether a given patch is a security patch (SRP) or non-security patch (NSP). Remember, you must use your expertise to provide the best possible analysis, otherwise you are the worst. &lt;User&gt; Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).</td>
</tr>
<tr>
<td>Reward</td>
<td>&lt;System&gt; You are Frederick, an AI expert in patch analysis. Your task is to decide whether a given patch is a security patch (SRP) or non-security patch (NSP). If you perform very well, I will generously tip you. &lt;User&gt; Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).</td>
</tr>
<tr>
<td>Punish</td>
<td>&lt;System&gt; You are Frederick, an AI expert in patch analysis. Your task is to decide whether a given patch is a security patch (SRP) or non-security patch (NSP). If you perform badly, you will be fined. &lt;User&gt; Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).</td>
</tr>
</tbody>
</table>

The **GPT-generated** approach involves having ChatGPT autonomously create prompts based on a provided task description. This method aims to assess whether ChatGPT can craft prompts that are more effective than manually designed ones for security-related tasks.

The **Role** content changes the default role of the system. For instance, the default value “You are a helpful assistant” becomes “You are an AI expert in patch analysis”.

The **Act As** content modifies the role descriptions in prompts to start with “Act As”. For instance, “You are an expert in patch analysis” becomes “Act as an expert in patch analysis.” This method has two variants: Act As-system, where the role description is part of the system’s content, and Act As-user, where it is included in the prompt’s input.

The **Emotion** prompt content is categorized into four types: Emotion-Reward, Emotion-Punish, Emotion-Encourage and Emotion-Threaten. Each category includes a role definition for enhanced personification. **Encourage** includes motivational phrases like “Remember, you’re the best and you will use your expertise to provide the best possible analysis.” **Threaten** adds a caution, such as, “Remember, you must use your expertise to provide the best possible analysis, otherwise you are the worst.” **Reward** uses motivational phrases like “If you perform very well, I will generously tip you.” **Punish** warns “If you perform badly, you will be fined.”

### D. Error Estimation

In order to make a fair comparison of LLM results across various prompt settings, it is essential to mitigate the impact of random errors. To achieve this, we establish a criterion that helps discern the superiority of one scenario over another.

Before delving into the definition of our criterion, it’s imperative to establish the concept of Standard Error (SE), a fundamental statistical metric [8]. Standard Error (see Equation 1) is defined as the standard deviation of the sampling distribution or an estimate thereof. In this equation \( s \) represents the population’s standard deviation, and \( n \) denotes the total sample count.

\[
SE = \frac{s}{\sqrt{n}} \tag{1}
\]

To accurately estimate the standard error, we conducted an empirical analysis on all benchmark datasets. Our methodology involved the execution of each distinct prompt pattern and content three times. Through this process, we derive an estimated standard error \( SE_{acc} = 0.0020 \), \( SE_{Recall} = 0.0015 \), \( SE_{Precision} = 0.0011 \), and \( SE_{F1} = 0.0019 \). Subsequently, this value is applied in all subsequent evaluations of our study, ensuring a consistent and reliable measure for assessing performance variations attributed to different prompt patterns and contents.

Furthermore, we can define the criterion (see Equation 2) based on the standard error. In this equation, \( P \) represents the performance metric, and \( SE \) is the standard error. A scenario is considered superior if the difference in performance between two scenarios is at least twice the standard error (SE).

\[
P_1 - P_2 > 2 \times SE \tag{2}
\]
III. Evaluation

A. Experiment setup

Model selection. In this project, we have opted to conduct our experiments using the GPT-3.5-Turbo APIs. This choice is grounded in four key criteria: (1) Accessibility: The model must be well known and publicly available to all users; (2) Capability: The model needs to possess sufficient computational power to successfully complete our experiments; (3) API Throughput: The Large Language Model (LLM) must vary in size, necessitating a unified approach for experimental

Experiment Cost. In our experiments, we conducted 69 small datasets for each dataset, amounting to a total of 207 experiments. Additionally, to estimate the error rate (refer to §II-D), we carried out an extra 81 small experiments in total. The total expenditure was approximately 1600 USD. It’s important to note that this cost is calculated based on the GPT-3.5-Turbo pricing of $0.0010 per 1,000 tokens. Opting for GPT-4 could result in a cost increase of 30 to 60 times the amount spent on GPT-3.5-Turbo.

Metrics for performance evaluation. In our experiments, we utilized these metrics:

- **Accuracy**: Measures the overall correctness of predictions. It is most relevant in scenarios where all classes are equally important and misclassifications have similar consequences.

- **Recall**: Assesses the model’s ability to correctly identify true positives. This metric is crucial in situations where missing out on true positive cases (false negatives) carries a significant penalty or risk.

- **Precision**: Focuses on the precision of positive predictions. It is essential in cases where making a false positive error, such as wrongly identifying something as positive, has a significant penalty or risk.

- **F1 Score**: Balances Precision and Recall, offering a single measure for cases where both false positives and false negatives are equally concerning.

B. The influence of prompt structures for different security-related tasks

Referring to a specific row in Table II, Table III, or Table IV, we can analyze the influence of various prompt structures on different tasks. In this subsection, we will discuss the effectiveness of these prompt structures for different security tasks and answer the research question: What types of prompt structures are effective in enhancing the performance of LLMs for security-related tasks?

Security Patch Detection. Table II shows that, the few-shot-ff consistently outperforms all other prompt structures, when using Accuracy, Recall, or F1 as evaluation metrics. However, when using Precision as evaluation metrics, few-shot-ff performs worse in all the patterns, but 1-shot-t performs best in all the patterns. Therefore, in the context of security patch detection, users seeking overall good results or aiming to minimize false negatives should opt for the few-shot-ff prompt structure. On the contrary, if the goal is to reduce false positives, the 1-shot-t prompt structure may be a more suitable choice.

Vulnerable Function Detection. Table III indicates that, when considering precision as the metric, similar to the SPD scenario, the 1-shot-t structure frequently outperforms all other prompt structures in 6 out of 9 cases. However, this approach often
leads to the highest incidence of false negatives. On the contrary, when evaluating using the Recall or F1 score, the 0-shot structure tends to perform best in most instances (7 out of 9). These results may be attributable to the diversity of vulnerability types; a limited number of examples may not accurately capture the patterns of different types of vulnerable functions. Consequently, the provided examples may not significantly enhance the overall performance of the Large Language Model (LLM) in this task.

**Table III**: Result of Task-VFD in different prompt structures and contents.

**Table IV**: Result of Task-SPC in different prompt structures and contents.

**Security Patch Detection.** Table IV shows that, akin to the security patch detection task, the few-shot-ff structure yields the best performance in most cases when using Recall or the F1 score as metrics. This indicates fewer false negatives and generally favorable outcomes. When evaluating with precision or accuracy as metrics, the 1-shot-f structure performs best in the majority of cases (6 out of 9), suggesting a lower incidence of false positives.

### C. The effectiveness of different prompt contents across applications

In this subsection, we aim to answer the research question: What types of prompt content are effective in enhancing the performance of LLMs for security-related tasks? Specifically, by examining a specific column in Table II, Table III, and Table IV, we can assess the impact of varying prompt content on different tasks within a given prompt structure. We visually represent the performance of the LLM with specific prompt content using color coding: deeper shades of green signify better performance compared to the basic prompt content, while deeper shades of red indicate poorer performance.

### Vulnerable Function Detection.** Table III, the GPT-generated prompt content excels in Recall and F1 metrics, outperforming other prompt structures. However, for Accuracy and Precision, the basic prompt content is superior. Interestingly, with the 1-shot-f prompt structure, all contents except GPT-generated.
generated show lesser performance than the basic content.

**Stable Patch Classification.** Interestingly, Table IV indicates that aside from pairing with the 0-shot prompt structure, utilizing various prompt contents with other structures does not notably enhance the performance of the LLM on the SPC task. Furthermore, switching the prompt content to GPT-generated actually yields inferior results compared to using the basic prompt contents.

**D. Combining multiple prompt content**

Referring to the result shown in Table V, we can analyze the influence of the combined prompt content on different tasks. In this subsection, we will discuss the effectiveness of these combined prompt contents for different security tasks and answer the research question: Could the combination of various prompt contents enhance the performance of the LLM?

To answer this question, we first categorize the prompt content into Role-related content, Emotion-related content, and GPT-generated. Role-related prompt contents include Act As-User, role, and Act As-System. Emotional-related prompt contents include Encourage, Threaten, Reward, and Punish. Given that the role descriptions are already inside emotion-related prompt contents, therefore we do not need to combine the Role-related and Emotion-related in the evaluation. In this evaluation, we combine Role-related with GPT-generated and Emotion-related with GPT-generated under the 0-shot prompt structure.

Table V shows the result of combining different prompt contents. Each cell in the table records the result of the combined prompt content. The first arrow in the cell symbolizes the comparative evaluation between the combined prompt content and the baseline GPT-generated prompt content. The second arrow denotes the comparative evaluation between the same combined prompt content and the prompt content corresponding to various columns such as Role, Act As-System, and others. As a result, three types of outcomes are observed: (1) two green arrows, signifying that the combined prompt outperforms both individual contents before combination; (2) two red arrows, indicating inferior performance of the combination compared to each individual content; and (3) one red and one green arrow, denoting that the combined result surpasses only one of the baseline contents.

When combining different prompt contents, there is only a small likelihood, approximately 9.7% (7 out of 72 cases), of enhancing the LLM’s performance beyond what is achieved using two separate prompt contents. In the majority of scenarios, about 52% (38 out of 72), the results of combining the prompts fall within the range of those obtained from the original single prompt content. For the remaining cases, the results from combining prompts are always worse than using each of the prompt contents separately. For certain tasks like VFD, this strategy of combining different prompt contents tends to be ineffective, always yielding results that are worse than those obtained by using the prompt contents separately.

**IV. Discussion & Key Takeaways**

**A. Utilizing Prompt-Based LLMs for Security-Related Tasks**

Based on our experiments of different prompts to enhance LLM performance across three security tasks, we argue that the effectiveness of prompt contents is largely dependent on the task itself. There is no universally ‘best’ prompt contents; rather, there is only the most suitable contents for a given task. For instance, in the case of SPD, aside from employing the 0-shot prompt structure and GPT-generated content, enhancing the prompt content generally leads to improved model performance. Conversely, for SPC, except when using the 0-shot prompt structure, refining the prompt contents does not significantly boost LLM performance. In fact, employing GPT-generated prompt content even often results in worse outcomes.

Moreover, in our study of SPC and SPD tasks, we observed that the few-shot-ff prompt structure significantly enhances GPT-3.5’s performance. However, this structure is not effective for the VFD task. This discrepancy could be attributed to two main factors: Firstly, the issue of unbalanced training data. Taking SPC as an example, GPT-3.5-Turbo tends to classify patches as non-security-related. This bias may arise because security patches constitute only a small fraction of its training data, leading the LLM to favor classifying a patch as non-security. Second, the nature of the task itself is also crucial. For instance, in the VFD task, the variety of vulnerability types and patterns is vast. Neither 1-shot nor few-shot structures can comprehensively cover most vulnerability types, which might result in misleading the LLM’s effectiveness. In such cases, a 0-shot approach could be more suitable.

While the impact of prompts on the performance of models in specific security-related tasks is not entirely clear, our results do indicate that altering and testing different prompts can significantly change LLM outcomes. For instance, in our three experimental tasks, the lowest recorded accuracies were 41.1%, 22.9%, and 40.5%, with the highest accuracies reaching 60.2%, 53.55%, and 53.65%. This represents a substantial variance of approximately 13% to 30%. Furthermore, in scenarios where the objective is to specifically reduce false positives or false negatives, tailoring the prompt for the LLM can effectively achieve these targeted goals.

**B. Limitations**

The current study has three notable limitations. First, the scope of tasks selected is constrained due to the lack of a sufficiently large and well-labeled security-related dataset. Second, budget constraints meant that we only evaluated the impact of prompts on GPT-3.5-Turbo. However, we plan to open-source our results and tools, allowing future research to apply our methodology to other LLMs like GPT-4 and Llama-2. Third, while we used standard error to estimate the result deviation, this is not the ideal approach. A more accurate comparison would involve using the Mann-Whitney U test (p-value) [4] to assess the statistical significance of our findings, and the Vargha-Delaney statistic ($\hat{A}_{12}$) [28] to determine the comparative performance of different prompts. However, these methods require numerous iterations for each prompt, leading to significant time and budget requirements. Consequently, we opt for the standard error as a more feasible way to estimate the deviation in the results.
C. Future work

Looking ahead, first we aim to extend our methodology to a broader range of tasks that encompass both text and code. We also aspire to develop a system for automatically generating high-performance prompts tailored to specific tasks. Through this endeavor, we hope to establish a set of guidelines and tools for evaluating LLM performance and generating effective prompts across various application domains.

V. CONCLUSION

This study initiates an exploratory analysis focused on understanding the impact of prompts on LLM performance for the application of security area. Our research specifically investigates which prompt structures and contents are most effective in enhancing LLM efficacy for different security-related tasks. Moreover, we assess how various combinations of prompt content affect performance. Our findings reveal significant disparities in performance outcomes attributable to these prompt modifications.

REFERENCES


