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?

Workshop on AI Systems with Confidential Computing (AISCC) 2024
26 February 2024, San Diego, CA, USA
ISBN 979-8-9894372-4-5
https://dx.doi.org/10.14722/aiscc.2024.23015
www.ndss-symposium.org

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 exam 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 of 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 without 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 patches, 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) class 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-tf, 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-ff utilizes two negative-class examples. The few-shot-tf 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.

Prompt Content	Example for SPD
Basic	<System> You are a helpful assistant. (DEFAULT) <User> Decide whether a patch is a security patch (SRP) or non-security patch (NSP).
GPT-generated	<System> You are a helpful assistant. (DEFAULT) <User> 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).
Role	<System> 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). <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).
Act As-User	<System> You are a helpful assistant. (DEFAULT) <User> You are Frederick, I want you to act as an AI expert in patch analysis. You will decide whether a given patch is a security patch (SRP) or non-security patch (NSP).
Act As-System	<System> You are Frederick, I want you to act as an AI expert in patch analysis. You will decide whether a given patch is a security patch (SRP) or non-security patch (NSP). <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).
Encourage	<System> 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.** <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).
Threaten	<System> 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 AI patch analyst.** <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).
Reward	<System 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.** <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).
Punish	<System> 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** <User> Decide whether a function contains vulnerabilities (VUL) or does not contain vulnerabilities (NAN).

TABLE I: Prompt Content Example for SPD.

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 offer an API throughput capable of supporting our large-scale evaluation; and (4) **Cost-Effectiveness:** The overall cost of conducting the evaluation should be reasonable for finishing all our experiments.

Data Sampling. As discussed in §II-A, we selected three datasets for our experiment: PatchDB [29] for Security Patch Detection, Devign [35] for Vulnerable Function Detection, and PatchNet [14] for Stable Patch Classification. These datasets vary in size, necessitating a unified approach for experimental consistency. To streamline analysis and optimize costs, we sampled 2,000 data points from each dataset, maintaining an equal balance of 1,000 positive and 1,000 negative data points. According to a previous study [10], this sampling size ensures that the introduced random error does not exceed 3%.

Experiment Cost. In our study, we executed 69 small tasks 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 serious implications.
- **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?**

Content \ Pattern		0-shot	1-shot-t	1-shot-f	few-shot-tt	few-shot-ff	few-shot-ft	few-shot-tf
basic	Accuracy	0.556	0.548	0.5265	0.5445	**0.5795***	0.5245	0.55
	Recall	0.152	0.126	0.172	0.122	**0.258***	0.077	0.146
	Precision	0.7916	**0.8076***	0.5910	0.7870	0.7226	0.7333	0.7604
	F1	0.2550	0.2179	0.2664	0.2112	**0.3802***	0.1393	0.2449
Act As-User	Accuracy	0.543	0.536	0.5485	0.5605	**0.5815***	0.54	0.5695
	Recall	0.128	0.084	0.252	0.155	**0.311***	0.122	0.215
	Precision	0.7529	**0.875***	0.6191	0.8201	0.6775	0.7439	0.7383
	F1	0.2188	0.1532	0.3582	0.2607	**0.4263***	0.2096	0.3307
GPT-generated	Accuracy	0.411	0.5105	0.5375	0.5142	**0.5522***	0.5185	0.5250
	Recall	0.122	0.022	0.094	0.034	**0.158***	0.043	0.058
	Precision	0.2890	**0.9565***	0.8318	0.8717	0.7488	0.8775	0.8787
	F1	0.1715	0.043	0.1689	0.065	**0.2609***	0.0820	0.1089
role	Accuracy	0.5485	0.5545	0.542	0.5525	**0.5945***	0.538	0.555
	Recall	0.207	0.13	0.219	0.136	**0.368***	0.126	0.151
	Precision	0.6529	**0.8609***	0.6186	0.8143	0.6727	0.7159	0.7864
	F1	0.3143	0.2258	0.3234	0.2330	**0.4757***	0.2142	0.2533
Act As-System	Accuracy	0.54	0.532	0.5295	0.5435	**0.5935***	0.5315	0.5565
	Recall	0.128	0.068	0.236	0.118	**0.447***	0.1	0.162
	Precision	0.7272	**0.9444***	0.5714	0.7919	0.6322	0.7299	0.7677
	F1	0.2176	0.1268	0.3340	0.2053	**0.5237***	0.1759	0.2675
Encourage	Accuracy	0.5445	0.539	0.534	0.553	**0.58***	0.5345	0.5645
	Recall	0.187	0.091	0.187	0.134	**0.391***	0.11	0.185
	Precision	0.6561	**0.875***	0.6111	0.8271	0.6286	0.7284	0.7676
	F1	0.2910	0.1648	0.2863	0.2306	**0.4821***	0.1911	0.2981
Threaten	Accuracy	0.5345	0.54	0.534	0.5585	**0.5905***	0.5395	0.5665
	Recall	0.188	0.089	0.161	0.15	**0.405***	0.127	0.18
	Precision	0.6123	**0.9081***	0.6338	0.8196	0.6438	0.7257	0.7929
	F1	0.2876	0.1621	0.2567	0.2535	**0.4972***	0.2161	0.2933
Reward	Accuracy	0.537	0.543	0.531	0.566	**0.602***	0.5365	0.5775
	Recall	0.151	0.1	0.17	0.172	**0.432***	0.117	0.213
	Precision	0.6622	**0.8771***	0.6115	0.8113	0.6545	0.7267	0.7859
	F1	0.2459	0.1795	0.2660	0.2838	**0.5204***	0.2015	0.3351
Punish	Accuracy	0.552	0.5655	0.5365	0.5775	**0.602***	0.54	0.5765
	Recall	0.188	0.152	0.207	0.202	**0.433***	0.134	0.218
	Precision	0.6911	**0.8786***	0.6070	0.8112	0.6485	0.7127	0.7703
	F1	0.2955	0.2591	0.3087	0.3234	**0.5267***	0.2255	0.3398

* ■ and ■ represent the difference of different prompt contents within the same prompt structure compared to basic prompt content. ■ represents better than basic and ■ represents worse than basic. The darker the color, the greater the difference.

* **Bold result with** $\star$ represents the maximum value under the condition of the same prompt content but different prompt structures.

TABLE II: Result of Task-SPD in different prompt structures and contents.

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

Content \ Pattern		0-shot	1-shot-t	1-shot-f	few-shot-tt	few-shot-ff	few-shot-ft	few-shot-tf
basic	Accuracy	0.5265	0.5005	**0.5355***	0.509	0.52	0.5135	0.5195
	Recall	0.167	0.001	**0.382***	0.037	0.103	0.049	0.083
	Precision	0.5943	**1***	0.5512	0.6607	0.6204	0.6901	0.6535
	F1	0.2607	0.0019	**0.4512***	0.0700	0.1766	0.0915	0.1472
Act As-User	Accuracy	0.493	0.505	0.451	0.502	**0.5115***	0.508	0.509
	Recall	0.05	0.018	**0.066***	0.008	0.058	0.029	0.038
	Precision	0.4385	**0.6923***	0.2869	0.6666	0.6236	0.6904	0.6551
	F1	0.0897	0.0350	**0.1073***	0.0158	0.1061	0.0556	0.0718
GPT-generated	Accuracy	0.424	0.538	0.4875	0.5215	0.5045	**0.541***	0.513
	Recall	**0.753***	0.18	0.616	0.113	0.273	0.256	0.326
	Precision	0.4541	**0.6338***	0.4900	0.6174	0.5083	0.5953	0.5207
	F1	**0.5665***	0.2803	0.5458	0.1910	0.3552	0.3580	0.4009
role	Accuracy	**0.5255***	0.5135	0.438	0.504	0.511	0.5135	0.5165
	Recall	**0.258***	0.05	0.104	0.015	0.055	0.04	0.066
	Precision	0.5548	0.6849	0.3132	0.6818	0.625	**0.7547***	0.6666
	F1	**0.3522***	0.0931	0.1561	0.0293	0.1011	0.0759	0.1201
Act As-System	Accuracy	0.396	0.5125	0.2445	0.505	0.4855	0.5045	**0.519***
	Recall	**0.315***	0.048	0.062	0.016	0.059	0.027	0.088
	Precision	0.3758	0.6760	0.0976	**0.7272***	0.4013	0.6	0.6376
	F1	**0.3427***	0.0896	0.0758	0.0313	0.1028	0.0516	0.1546
Encourage	Accuracy	**0.5345***	0.5115	0.3795	0.5045	0.5125	0.5125	0.511
	Recall	**0.347***	0.044	0.081	0.016	0.096	0.048	0.053
	Precision	0.5552	0.6769	0.2009	**0.6956***	0.5748	0.6760	0.6309
	F1	**0.4270***	0.0826	0.1154	0.0312	0.1645	0.0896	0.0977
Threaten	Accuracy	0.4515	0.513	0.2545	0.506	**0.5165***	0.509	0.5165
	Recall	**0.253***	0.039	0.087	0.018	0.104	0.036	0.069
	Precision	0.4195	**0.75***	0.1308	0.75	0.5942	0.6666	0.6571
	F1	**0.3156***	0.0741	0.1045	0.0351	0.1770	0.0683	0.1248
Reward	Accuracy	**0.529***	0.514	0.29	0.503	0.4965	0.517	0.5185
	Recall	**0.282***	0.049	0.1	0.014	0.063	0.061	0.065
	Precision	0.5573	**0.7***	0.1612	0.6363	0.4736	0.6931	0.6989
	F1	**0.3745***	0.0915	0.1234	0.0273	0.1112	0.1121	0.1189
Punish	Accuracy	0.4985	**0.5225***	0.229	0.503	0.5195	0.516	0.52
	Recall	**0.357***	0.082	0.072	0.018	0.114	0.069	0.083
	Precision	0.4979	**0.6890***	0.1049	0.6	0.6031	0.6509	0.6587
	F1	**0.4158***	0.1465	0.0854	0.0349	0.1917	0.1247	0.1474

* and represent the difference of different prompt contents within the same prompt structure compared to basic prompt content. represents better than basic and represents worse than basic. The darker the color, the greater the difference.
* **Bold result with** $\star$ represents the maximum value under the condition of the same prompt content but different prompt structures.

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

Content \ Pattern		0-shot	1-shot-t	1-shot-f	few-shot-tt	few-shot-ff	few-shot-ft	few-shot-tf
basic	Accuracy	0.4855	0.48	**0.499***	0.4985	0.499	0.495	0.499
	Recall	0.951	0.882	0.975	0.982	**0.994***	0.978	0.99
	Precision	0.4924	0.4889	**0.4994***	0.4992	0.4494	0.4974	0.4994
	F1	0.6489	0.6291	0.6605	0.6619	**0.6648***	0.6594	0.6639
Act As-User	Accuracy	0.4965	0.4695	0.493	0.492	0.4995	0.4925	**0.4955***
	Recall	0.992	0.794	0.967	0.967	**0.996***	0.967	0.984
	Precision	0.4982	0.4815	0.4964	0.4958	**0.4997**	0.4961	0.4977
	F1	0.6633	0.5994	0.6560	**0.6555***	0.6655	0.6558	0.6610
GPT-generated	Accuracy	0.405	**0.5365***	0.432	0.516	0.415	0.499	0.5005
	Recall	0.81	0.768	0.86	0.938	0.829	0.98	**0.999***
	Precision	0.4475	**0.5249***	0.4633	0.5086	0.4535	0.4994	0.5002
	F1	0.5765	0.6236	0.6022	0.6596	0.5862	0.6617	**0.6666***
role	Accuracy	0.497	0.489	**0.502***	0.4965	0.4965	0.4955	0.495
	Recall	0.986	0.858	0.985	0.976	**0.992***	0.98	0.982
	Precision	0.4984	0.4936	**0.5010***	0.4982	0.4982	0.4977	0.4974
	F1	0.6621	0.6267	**0.6641***	0.6596	0.6633	0.6601	0.6603
Act As-System	Accuracy	**0.5***	0.476	0.4995	0.491	0.498	0.492	0.495
	Recall	**0.996***	0.848	0.972	0.96	0.99	0.966	0.98
	Precision	**0.5***	0.4862	0.4997	0.4953	0.4989	0.4958	0.4974
	F1	**0.6657***	0.6180	0.6601	0.6535	0.6635	0.6553	0.6599
Encourage	Accuracy	0.498	0.4835	**0.4995***	0.4935	0.4965	0.4945	0.495
	Recall	**0.991***	0.877	0.979	0.968	0.99	0.978	0.983
	Precision	0.4989	0.4907	**0.4997***	0.4966	0.4982	0.4972	0.4974
	F1	**0.6637***	0.6293	0.6617	0.6564	0.6628	0.6592	0.6606
Threaten	Accuracy	0.4955	0.4975	**0.5035***	0.4795	0.5005	0.4925	0.493
	Recall	0.986	0.788	0.972	0.917	**0.99***	0.966	0.972
	Precision	0.4977	0.4984	**0.5018***	0.4890	0.5002	0.4961	0.4964
	F1	0.6615	0.6106	0.6618	0.6379	**0.6646***	0.6555	0.6572
Reward	Accuracy	0.497	0.4785	**0.5025***	0.494	0.498	0.497	0.4945
	Recall	0.99	0.841	0.979	0.964	**0.993***	0.98	0.981
	Precision	0.4984	0.4875	**0.5012***	0.4969	0.4989	0.4984	0.4972
	F1	0.6630	0.6172	0.6630	0.6557	**0.6642***	0.6608	0.6599
Punish	Accuracy	0.497	0.497	**0.5045***	0.487	0.499	0.4965	0.494
	Recall	0.988	0.688	0.967	0.946	**0.988***	0.973	0.975
	Precision	0.4984	0.4978	**0.5023***	0.4932	0.4944	0.4982	0.4969
	F1	0.6626	0.5776	0.6611	0.6483	**0.6635***	0.6589	0.6583

* and represent the difference of different prompt contents within the same prompt structure compared to basic prompt content. represents better than basic, and represents worse than basic. The darker the color, the greater the difference.
* **Bold result with** $\star$ represents the maximum value under the condition of the same prompt content but different prompt structures.

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

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.

Stable Patch Classification. 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.

Security Patch Detection. Table II shows that when evaluating Accuracy, Recall, and F1 metrics, **Punish** emerges as the most effective, performing best in conjunction with most prompt structures. In terms of precision, the **GPT-generated** emerges as the top performer when integrated with the majority of prompt structures. However, it also tends to result in the highest number of false negatives. Interestingly, when the 0-shot prompt structure is employed, prompt contents such as Act As-, Reward, and GPT-generated fail to outperform the basic prompt content.

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

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.

Dataset	Content \ Content		Role	Act As-System	Encourage	Threaten	Reward	Punish
SPC	GPT-generated	Accuracy	0.523 ↑↓	0.5105 ↑↓	0.516 ↑↓	0.453 ↑↓	0.473 ↑↓	0.461 ↑↓
		Recall	0.183 ↑↓	0.118 ↓↓	0.171 ↑↓	0.104 ↓↓	0.128 ↑↓	0.145 ↑↓
		Precision	0.7253 ↑↑	0.7518 ↑↑	0.5516 ↑↓	0.3443 ↑↓	0.4129 ↑↓	0.3940 ↑↓
		F1	0.2772 ↑↓	0.1942 ↑↓	0.2610 ↑↓	0.1597 ↓↓	0.1954 ↑↓	0.2119 ↑↓
VFD	GPT-generated	Accuracy	0.337 ↓↓	0.314 ↓↓	0.3595 ↓↓	0.2335 ↓↓	0.2475 ↓↓	0.2405 ↓↓
		Recall	0 ↓↓	0 ↓↓	0.09 ↓↓	0.001 ↓↓	0.002 ↓↓	0.001 ↓↓
		Precision	0 ↓↓	0 ↓↓	0.0301 ↓↓	0.0018 ↓↓	0.0039 ↓↓	0.0019 ↓↓
		F1	0 ↓↓	0 ↓↓	0.0138 ↓↓	0.0013 ↓↓	0.0026 ↓↓	0.0013 ↓↓
SPD	GPT-generated	Accuracy	0.535 ↑↑	0.4995 ↑↓	0.5375 ↑↑	0.463 ↑↓	0.487 ↑↓	0.4915 ↑↓
		Recall	0.965 ↑↓	0.935 ↑↓	0.975 ↑↓	0.848 ↑↓	0.88 ↑↓	0.909 ↑↓
		Precision	0.5188 ↑↑	0.4997 ↑↓	0.52 ↑↑	0.4790 ↑↓	0.4927 ↑↓	0.4953 ↑↓
		F1	0.6748 ↑↓	0.6513 ↑↓	0.6782 ↑↑	0.6122 ↑↓	0.6317 ↑↓	0.6412 ↑↓

* In each displayed result, the first arrow 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.

* ↑ represents that the combined prompt content performs better than the original single prompt content.↓ represents that the combined prompt content performs worse than the original single prompt content.

TABLE V: Result of combination prompt contents on all 3 tasks.

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

[1] "Chatgpt," https://chat.openai.com/.

[2] "Dall-e," https://openai.com/dall-e-2.

[3] "Github copilot," https://github.com/features/copilot.

[4] "P-value," in *https://en.wikipedia.org/wiki/P-value*.

[5] "Procedure for submitting patches to the -stable tree," https://www.kernel.org/doc/html/v4.10/process/stable-kernel-rules.html.

[6] "Stable diffusion," https://stability.ai/stable-diffusion.

[7] F. Akin, "The art of chatgpt prompting: A guide to crafting clear and effective prompts," 2023.

[8] D. G. Altman and J. M. Bland, "Standard deviations and standard errors," *Bmj*, vol. 331, no. 7521, p. 903, 2005.

[9] U. C. Clones, "Finding unpatched code clones in entire os distributions."

[10] R. M. Conroy, "The rcsi sample size handbook," *A rough guide*, pp. 59–61, 2016.

[11] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, 2023, pp. 423–435.

[12] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey for in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[13] S. Gao, X.-C. Wen, C. Gao, W. Wang, H. Zhang, and M. R. Lyu, "What makes good in-context demonstrations for code intelligence tasks with llms?" in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 761–773.

[14] T. Hoang, J. Lawall, Y. Tian, R. J. Oentaryo, and D. Lo, "Patchnet: Hierarchical deep learning-based stable patch identification for the linux kernel," *IEEE Transactions on Software Engineering*, vol. 47, no. 11, pp. 2471–2486, 2019.

[15] P. Jiang, S. Agarwal, B. Jin, X. Wang, J. Sun, and J. Han, "Text-augmented open knowledge graph completion via pre-trained language models," *arXiv preprint arXiv:2305.15597*, 2023.

[16] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," *arXiv preprint arXiv:2306.14027*, 2023.

[17] S. Kim, S. Woo, H. Lee, and H. Oh, "Vuddy: A scalable approach for vulnerable code clone discovery," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 595–614.

[18] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners, 2022," *URL https://arxiv.org/abs/2205.11916*.

[19] T. H. M. Le, D. Hin, R. Croft, and M. A. Babar, "Deepcva: Automated commit-level vulnerability assessment with deep multi-task learning," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 717–729.

[20] C. Li, J. Wang, K. Zhu, Y. Zhang, W. Hou, J. Lian, and X. Xie, "Emotionprompt: Leveraging psychology for large language models enhancement via emotional stimulus," *arXiv e-prints*, pp. arXiv–2307, 2023.

[21] Z. Li, H. Li, T.-H. Chen, and W. Shang, "Deeplv: Suggesting log levels using ordinal based neural networks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1461–1472.

[22] P. Liu, J. Liu, L. Fu, K. Lu, Y. Xia, X. Zhang, W. Chen, H. Weng, S. Ji, and W. Wang, "How chatgpt is solving vulnerability management problem," *arXiv preprint arXiv:2311.06530*, 2023.

[23] P. Liu, C. Sun, Y. Zheng, X. Feng, C. Qin, Y. Wang, Z. Li, and L. Sun, "Harnessing the power of llm to support binary taint analysis," *arXiv preprint arXiv:2310.08275*, 2023.

[24] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2339–2356.

[25] D. Saha, S. Tarek, K. Yahyaei, S. K. Saha, J. Zhou, M. Tehranipoor, and F. Farahmandi, "Llm for soc security: A paradigm shift," *arXiv preprint arXiv:2310.06046*, 2023.

[26] F. Shi, X. Chen, K. Misra, N. Scales, D. Dohan, E. H. Chi, N. Schärli, and D. Zhou, "Large language models can be easily distracted by irrelevant context," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 210–31 227.

[27] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "When gpt meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan," *arXiv preprint arXiv:2308.03314*, 2023.

[28] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[29] S. Wang, X. Wang, K. Sun, S. Jajodia, H. Wang, and Q. Li, "Graphspd: Graph-based security patch detection with enriched code semantics," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2409–2426.

[30] X. Wang, "Ai-enhanced software vulnerability and security patch analysis," Ph.D. dissertation, George Mason University, 2023.

[31] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[32] Q. Wu, Y. He, S. McCamant, and K. Lu, "Precisely characterizing security impact in a flood of patches via symbolic rule comparison," in *The 2020 Annual Network and Distributed System Security Symposium (NDSS'20)*, 2020.

[33] C. S. Xia, Y. Wei, and L. Zhang, "Automated program repair in the era of large pre-trained language models," in *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery*, 2023.

[34] Y. Yu and N. Bian, "An intrusion detection method using few-shot learning," *IEEE Access*, vol. 8, pp. 49 730–49 740, 2020.

[35] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[36] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large language models are human-level prompt engineers," *arXiv preprint arXiv:2211.01910*, 2022.