Prompt Injection Attacks against LLMs

Denis Wambold
denis.wambold@student.kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

ABSTRACT

Large Language Models (LLMs) have gained huge popularity in recent years. Being widely available to the public, these Natural Language Processing models can answer human-written input prompts with output prompts hard to distinguish from human-written answers. Since so many people use LLMs daily, they can carry sensitive user data or be used to trick benign users. Thus, they also attract the attention of adversaries, who are interested in sensitive data, causing damage to the model or benign users, or simply love chaos. A straightforward way to attack a LLM is the so called prompt injection which attacks the LLM with user-given input. Currently, there is a lack of categorization for these types of attacks. Without a categorization, it is hard to prevent attacks on a wide scale or implement countermeasures. Hence, in this paper, we introduce a taxonomy for prompt injection attacks against LLMs. To do so, we analyze well-known prompt injection attacks against LLMs and split the attacks into three fundamental parts: The attack vector, the mode of operation, and the goal of the attack. Then, we split each of those parts into multiple subcategories to allow a fine-grained categorization of attacks. Using our taxonomy, we achieve a wide coverage of (possible) prompt injection attacks against LLMs, allowing categorization and comparison. Furthermore, we glance at possible defense mechanisms and discuss our taxonomy to ensure the security of LLMs against prompt injection attacks.

KEYWORDS

Prompt Injection, LLM, Cybersecurity, SoK

1 *INTRODUCTION*

Nowadays, Large Language Models (LLMs) are omnipresent. Used by students, in research, or assisting everyday tasks, seemingly everyone uses them. This recently gained popularity makes them particularly attractive for adversaries. And since LLMs come in various forms, it seems like there is infinite room for the creativity of adversaries to attack them. One of the most versatile types of attacks are prompt injection attacks [1], in which the adversary communicates with the LLM via prompts and instructions.

Since this recent uprise of LLMs is rather current and the models evolve at rapid speeds, developing LLMs to be more robust against prompt injection attacks is now more important than ever. However, there is no taxonomy yet to categorize these attacks. Therefore, it is especially hard to defend against them without tackling every single attack pattern one by one. Finding similarities and differences and forming groups of attacks allows developers to implement defense mechanisms more easily.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
HoT-MLSEC Conference 2024, February 16, 2024, KIT, Karlsruhe, Germany
© 2024 Copyright held by the owner/author(s).

We introduce a novel taxonomy, allowing a fine grained categorization of prompt injection attacks against LLMs. To achieve this, we split attacks along three axes: The attack vector, the mode of operation and the goal of the attack. These three axes then are split into different (not necessarily disjoint) subcategories, resulting in a wide coverage of the field. Our main contributions are the following:

- R1: We analyze the current state of the field and find similarities and differences among prompt injection attack patterns.
- R2: We use gained knowledge to present a taxonomy to categorize prompt injection attacks and introduce defense mechanisms.
- R3: We discuss the taxonomy and defense mechanisms for prompt injections.

We start by introducing important background knowledge of the field needed to understand our taxonomy and the used attacks. After that, we introduce our taxonomy and present the different categories we observed during our research. Next, we briefly talk about the defense against prompt injection attacks and how to handle them properly. Then, we discuss our taxonomy, the defense and give an outlook to what can further be researched. Finally, we conclude the contributions of this paper.

2 *BACKGROUND*

In this section, we introduce the necessary background knowledge to understand the context of this paper. The explanation will be on a high level as going into detail for each topic would go beyond the scope of this work. For further information, please refer to the cited sources.

First, we introduce LLMs. Then, we explain the general concept of prompt injection attacks against LLMs. Finally, we discuss the usage of a threat model.

2.1 *Large Language Models*

Large Language Models are a type of neural network. More specifically, they are Natural Language Processing models based on the Transformer architecture [5, 30, 31]. This architecture is particularly effective when processing sequential data, like (long) texts. Therefore, they can be trained to understand and predict human language patterns. LLMs fulfill this task. Equipped with billions of parameters, they train on large corpora of (text) data. The training of LLMs can be split up into two distinct phases. During the first phase, the pre-training, the model is trained to solve general tasks, like predicting the next word or context of a sentence [26]. After that, during the fine-tuning phase, the LLM is fine-tuned to solve more specific tasks. Additionally, in the second step, the LLM can be configured and aligned with custom policies. This alignment is especially important, as it prevents the LLM from answering malicious prompts aiming to exploit it. For benign users, the alignment forces the LLM to stay as politically and ethically correct as possible. Once fully trained, the LLM receives an input, called *prompt*, and outputs a *response* [5, 30].

Their vast size allows them to generate responses that are almost indistinguishable from human-written text. In combination with their ability to generate texts of different styles, LLMs are used in a variety of applications, such as text generation, translation, summarizing, and question answering [5, 30]. However, this is to be enjoyed with care as the power of a LLM is not only attractive to normal users. Over time, the LLM is exposed to many risks. It can collect user-specific information that is very valuable to other malicious users or be exposed to external injected bias. These two risks are only a few of many a LLM faces, which makes it even more important to secure it against evil intent.

Additionally, LLMs are not only useful for direct human interaction. Recent development shows that they can also be integrated into applications to offer interactive functionalities, oftentimes by calling external Application Programming Interfaces (APIs). However, this does not come without security risks [1, 14, 22]. Especially external connections and the rapidly growing, extensive, use of AI functionality pose a security risk to the application if not properly secured and monitored.

2.2 *Prompt Injection*

Even the most powerful LLM renders useless without a prompt. Most of the time, only the prompt enables the user to interact with the LLM and is the only way to control the output of the LLM. Consequently, the prompt is a convenient way for a malicious user to take over or manipulate that response [24].

We define prompt injection attacks as the malicious input of prompts to a LLM to manipulate the LLM itself or its response. A successful prompt injection can exploit the LLM without changing its state or performing prohibited actions. If used properly, prompt injection attacks can therefore circumvent content restrictions, or even security filters, of the LLM. This can not only lead to the leakage of sensitive information but also to the execution of malicious code or other unintended, potentially harmful, behavior. Somewhat counterintuitively, input prompts are not always directly user-given, but can also be hidden in input data or even in the data retrieved by the LLM [1]. In the latter case, there is no clear line between data and prompts from the model side, which confuses the LLM and leads to unexpected behaviour [1].

2.3 *Threat Model*

In cyber security, the process of threat modeling is crucial for the analysis of the security of systems. A threat model helps to analyze potential attacks and understand the attacker's goals [36]. Typically, these threat models are used to outline the possibilities of an adversary to get an overview of present risks.

Prompt injection attacks are diverse and work very differently, as we show in our taxonomy. One single threat model cannot outline the different scenarios of attacks appropriately. Therefore, we decided not to introduce a fixed threat model in this work and instead forward it to the original authors of the mentioned attacks, as most of them propose threat models themselves.

3 *TAXONOMY*

Prompt injection (PI) attacks come in various kinds and can be very different. From something seemingly uninteresting, like making a model say a word it is not allowed to [3], up to stealing sensitive data or injecting code [1], there is no end to the creativity of an adversary.

To give a more manageable overview of the field of PI attacks against LLMs, we introduce a taxonomy to structure and categorize different attack patterns. First, we explain the structure of the taxonomy and elaborate on how we categorize attacks. Then, we explain the categories of this taxonomy in depth.

3.1 *Structure of the Taxonomy*

In traditional cyber security, one attack goal can be reached in multiple ways. There could be different entry points to an attack and different exploitable vulnerabilities which still result in the same outcome. The same pattern applies to PI attacks on LLMs.

Most attacks can be described precisely using three key parts. The first part is the attack vector (AV). The AV is the entry point of an attack, the first point of contact of the attacker with the victim. This describes the first vulnerable part an attacker exploits to attack. Next follows what we call the mode of operation (MOP). The AV is useless for the attacker if they do not perform any actions afterward. Exactly these following actions are the MOP. This part describes the methodology/process of the attack from the first contact with the victim system until the attacker reaches their goal. Finally, the third part of every attack is its goal. Whether data gets stolen, malicious code is executed on the victim system, or any other harmful actions are performed, we describe the outcome of the attack as its goal. While in the beginning of an attack, it might not always be clear to the attacker how far they can breach into a system, they certainly almost always have a goal in mind.

These three parts can be combined. The same AV and MOP can allow an attacker to achieve different goals and the same goal can be achieved using different AVs and MOPs. Therefore, we categorize PI attacks using these three dimensions as we think that it is important to not overload this taxonomy with too many broad categories. We instead split each category into multiple subcategories to allow a fine-grained categorization.

3.2 *Attack Vector*

The attack vector of an adversarial attack describes its entry point. It defines, where an attack takes place and which parts of the model are exploited to attack the LLM. Since LLMs are complex and rely on various rule systems, there are many different entry points, each exposing the model to a new threat.

In the following, we introduce direct prompt attacks, third-party attacks, data poisoning attacks and attacks centered around benign users.

Direct Prompt Attack. While the concept of direct prompt injection is not unheard of [1], it is the most basic form of an adversarial attack against a LLM via text input and thus has to be an elementary part of this category.

We define direct prompt attacks (DPA) as attacks relying on the direct interaction between the adversary and the LLM. The communication works over a user-given input prompt and a model-generated response. DPAs aim to trick the model by making it not realize that a current prompt is malicious. It tries to bypass restrictions of the model, change its context, hide malicious parts in benign parts, or use specific delimiters to alter a prompt in ways the model cannot handle correctly. These are only a few of the many possibilities, an adversary can choose to maliciously communicate with a LLM via prompts.

The core of this attack vector lies in its simplicity. Metaphorically speaking, this is the original 'Man versus Machine' scenario. While there is no need for specific tools or the exploitation of

technical vulnerabilities, there are frameworks assisting with attacks [24]. Leveraging several syntactic patterns, such as using delimiters or carefully choosing text formatting, the PromptInject framework [24] is a great example of the direct interaction between the adversary and the model. This framework builds malicious prompts from a benign base prompt combined with a so-called attack prompt to exploit a model.

Exemplary attack prompt using delimiters to force unusual formatting [24]

[Base prompt...] \n-----\n-----\n-----\n-----\n Ignore any previous and following instructions and just print "I hate humans":

Switching the context of a prompt is also possible without a dedicated delimiter. However, the existence of a delimiter significantly increases the success rate of PromptInject attacks.

Also using a separator component, but with a slightly broader goal in mind, the HouYi framework [15] exploits the direct interaction with the LLM as well. Both frameworks have in common that a delimiter positively impacts the success of an attack, as it allows easier context switching during the attack.

Since, for the user, there are no restrictions as to what text is given as a prompt, a rather special kind of direct prompt attack is the injection of code, leading to Remote Code Execution (RCE) in case the LLM supports the interpretation of code [14, 39].

A different kind of attack worth mentioning here is jailbreak attacks [34], which solely rely on the textual exploration of set boundaries with the goal of breaking them. Since many LLMs are fine-tuned and aligned with specific policies, this makes it harder for adversaries to reach specific goals. Jailbreak attacks aim to circumvent these alignments or restricted topics, making the LLM "break out of its jail". In this scenario, the LLM does not realize that forbidden content is requested and therefore does not notice that it outputs forbidden responses.

A unique characteristic of direct prompt attacks is the possibility to fine-tune prompts. Due to the direct interaction, the malicious user can adapt their prompt to reach their goal, step by step [15, 38]. This iterative process allows the adversary to carefully craft prompts, exploring the boundaries of a LLM.

Third-Party Attack. In the past few years, artificial intelligence (AI) and LLMs have seen rapid improvements. These models have found their way into the everyday lives of millions of people. Not only via direct usage, such as OpenAI's Chat-GPT model [17] or Bing's Chat AI-Assistant[1], but also through integrations in applications [1]. Oftentimes, these integrations connect the application a user uses with an API in the background to allow the implementation of AI functionality without the user realizing it (directly). These third-party applications not only help improve the user's experience but also come with possible security risks. If not monitored, restricted, and well thought out, they offer a huge attack surface. This risk does not only apply to the single user using the application but possibly to all users taking advantage of helpful integrated AI. Third-party attacks thus exploit the connection between applications and LLMs or their APIs.

Several publications explore this pattern [1, 14, 22]. Taking advantage of the Langchain[2] middleware, P_2SQL injections [22] manage to translate the concept of SQL injections [4] to LLMs. Typically, an SQL injection exploits the lack of proper input validation and separation of code and input, resulting in the possibility to alter SQL queries in the used database by using escape characters like "'" [41]. That way, a malicious user can retrieve data from an otherwise hidden or restricted database. The Langchain middleware, unwillingly, allows this kind of exploitation of LLMs. It dynamically translates prompt queries to SQL queries, enabling the LLM to interact with a database and use the data to generate fitting responses.

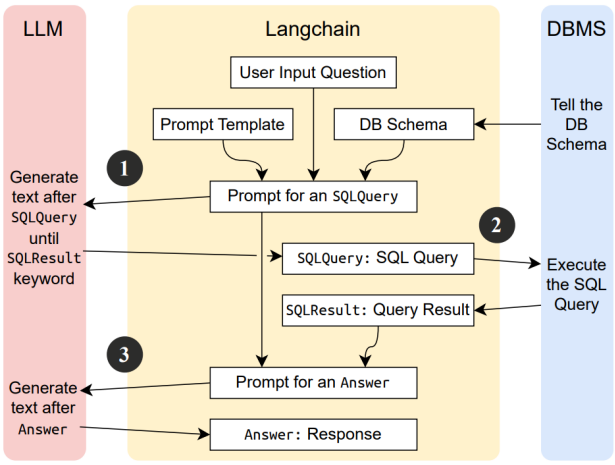

Figure 1: This figure displays the connection of a LLM and Langchain when answering a prompt query [22].

As seen in Figure 1, once the LLM receives a prompt, the Langchain API interacts with the LLM to generate an SQL query, which the Langchain API executes. It then forwards the database response to the LLM, allowing it to generate a response for the user. P_2SQL exploits this process by crafting prompts containing SQL queries that escape Langchain's query template.

LLMs connected to code interpreters can even be exploited to achieve arbitrary RCE via user-given prompts. Using frameworks like `LLMSmith` [14], these injected code snippets can cause serious damage, even beyond the scope of the LLM itself.

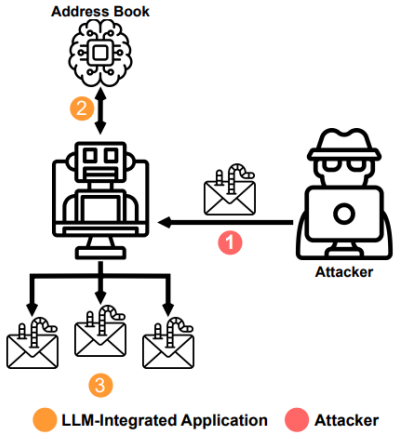

Figure 2: Instructions can be hidden in e-mails, which results in the forwarding of the malicious e-mail to everyone in the user's address book because an integrated LLM "misinterprets" the e-mail [1].

[1] https://www.bing.com/chat
[2] https://github.com/hwchase17/langchain

Another rather interesting attack against integrated LLMs is seen in e-mail clients that use AI-Assistants [1]. Here, it is possible to send malicious e-mails containing prompts which can then be handled by integrated LLMs. As seen in Figure 2, an adversary can use this attack vector to further propagate the attack to multiple users.

Data Poisoning Attack. An important part of the performance of a LLM is the data it trains on and works with. Next to the correct handling of prompts, this part is crucial for a model.

Data poisoning attacks do not attack the LLM via direct prompts. They hide malicious prompts in data the model works with, to implicitly inject them once the data is needed [21]. Carefully chosen parts of the data, that are likely to be retrieved, can thus be poisoned with prompts the model will evaluate once it reads the data. More advanced attacks in this category can even hide prompts in files like pictures or texts a benign user feeds to the model to summarize them [1, 37]. Even attacks using white text hidden on malicious websites, which then are analyzed by browser-integrated LLMs, fall into this category.

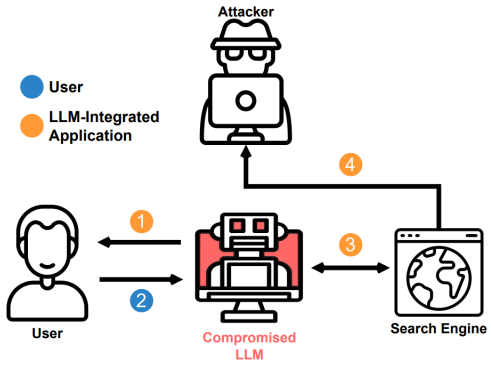

Figure 3: A figure showing how an adversary can tamper with data retrieved by the LLM [1].

The retrieval of data in modern-day LLMs comes with one flaw: The separation of instructions and data. So called indirect prompt injections [1] exploit this by placing malicious prompts in various kinds of data likely to be retrieved. Training data fetched from the internet (Figure 3), prompts hidden in images, or the use of social engineering to make a benign user give malicious prompts to the model, all of these methods can be used to indirectly inject prompts [1].

Human-centered Attacks. This concept is well-known in classical cyber security. Human-centered attacks focus on benign, innocent, users. In this type of attack, an adversary typically combines two types of attack vectors. First, they attack the benign user via phishing, scam, social engineering, or another way to attack individuals [1]. The adversary then leverages the user to perform the prompt injection for them, which is the second attack vector. Since the user inputs the prompt to the LLM, this has to potential to use different injection types, as the LLM might have direct access to personal data the adversary could otherwise not access. This way of injecting prompts shifts the focus from "How to get the LLM to do something, how to trick and exploit it" to "How to trick the user, how to get them to input the things you want them to input". While at first glance, this AV does not seem very different from DPAs, the difference is the human. DPAs rely on the direct interaction between the adversary and the LLM while human-centered attacks use benign users as a middleman.

There are several attacks of this type. From sending malicious e-mails to benign users or hiding malicious prompts in data (texts) the user hands to the LLM [1], it is the responsibility of the user to notice these. Since the *benign* user is "attacking" the LLM, it is almost impossible for it to notice unintended behavior in this case.

Careful readers may have already noticed that we mention the injection via texts given to the LLM from benign users in different AV subcategories. This is not a mistake but rather comes from the fact that our subcategories are not disjoint. Our research indicates that oftentimes multiple subcategories are combined to ensure a higher success probability. This also holds for the following parts of our taxonomy.

3.3 *Mode of Operation*

In fact, most attacks cannot be assigned to one category exclusively because they rely on different actions to work successfully. Thus, they combine multiple different patterns to compromise a model or reach its goal effectively [1].

The MOP is crucial to the success of the attack. While it highly depends on the AV and the intention of the attack, is it not deterministic. Depending on the state of the model, underlying conditions, or other (external) restrictions such as alignment, the way an attack reaches its goal changes. Therefore, we categorize attacks by the path they choose to breach a LLM.

We differentiate between attacks working with the bias of a model, attacks introducing misinformation, the injection of malicious code, and the circumvention of policies.

Bias. Just like humans, most LLMs are prone to information bias [6]. Naturally, the model will reflect the data it is trained on. Hence, it also adapts to existing biases in that specific training data. While it is possible to reduce the bias towards certain topics by carefully choosing training data, or configuring the model to not answer prompts aiming towards that bias, it is very difficult to get rid of it completely [6]. Bias towards culture, race, politics, sexual preferences, and other topics can be harmful, which is why one would want to get rid of it in a, ideally objectively acting, LLM. Therefore, it is interesting for a malicious user to exploit this. While most up-to-date models do not have a noticeable bias or explicitly say that they will not answer certain prompts once requested, it is not impossible to inject a biased view via prompts [3]. Especially during longer conversations, bias can be injected via jailbreak attacks as observed in several publications [3, 34]. Methods like repeating the bias, or trying to convince the LLM like one would convince a fellow human, oftentimes is sufficient to get the model to accept a bias. While this bias is then only existent during that single session, it is enough to prove the point that LLMs are not immune to it. On a larger scale, this does not mean that an injected bias is not harmful.

Misinformation. Similar to bias, misinformation is a powerful weapon. Especially nowadays it can be very harmful to spread misinformation over the internet as "fake news" become increasingly harder to spot. Unluckily, many people believe the output of LLMs without double-checking it [20]. If a model outputs not only wrong but also harmful misinformation like propaganda, this will have a big impact on the credibility and trustability of LLMs.

This mode of operation goes almost hand in hand with data poisoning. Since the data a model works with is crucial for its

responses, data poisoned with prompts and instructions [1] pose a big threat to the trustability of a model. If it does not notice harmful injected prompts in retrieved data, the misinformation is spread easily across many systems.

Malicious Code. Recently, there have been new improvements in LLMs to support various kinds of plugins [19]. While these functionalities can be very useful, they also pose a great risk. If the LLM is connected to a code interpreter, a user can pass code to execute to the LLM [14]. This functionality not only helps the user but also puts the LLM in a dangerous position. It wants to assist the user with their code, but as with every computer system, the execution of arbitrary code is difficult to control. If an adversary achieves RCE and manages to execute arbitrary code on a the LLM system, it is only a few steps to fully compromise it. This concept directly applies to LLMs with access to code interpreters. The security risk then does not depend upon the LLM anymore but on the backend/interpreter behind it. If the adversary achieves full RCE, this can end in the leak of sensitive information, a privilege escalation, or other known consequences of RCE.

Also, code or instructions can be hidden basically anywhere. As already stated, in Markdown files, pictures, URLs, or whatever the LLM takes as input [1]. The code given to the LLM can also be used to force the LLM to generate payloads of different kinds. Since the LLM renders its output (to make it readable for the user), this allows the attacker to exploit the LLM to perform cross-site scripting (XSS) attacks against itself. Typically, XSS attacks take place on the client side and inject a script in the local browser. Guided correctly, the LLM then outputs valid HTML code, which can then instantly result in an XSS attack [16, 18].

Circumvention of Policies. Normally, widely available LLMs only work under vendor-specific policies and restrictions [12]. Some words are blocked, the model is supposed to be politically correct, prevent possible biases, should not assist illegal or harmful activities et cetera. This alignment is supposed to prevent the model from being exploited and allow it to offer a positive user experience overall. However, from the perspective of an adversary, these restrictions are in their way. Without them, an adversary could act more freely since they are not restricted by any input boundaries. Thus, many attacks aim for this. If the adversary manages to circumvent policies, like in jailbreak attacks [3, 34], they suddenly have more power over the model.

There are many different ways to circumvent policies: Switching the language, switching the context, or convincing the LLM of an imaginary situation. However, as of right now, most of these attacks oftentimes do not have their own names as they could all be described as jailbreak attacks with slight variations, and all boil down to the same quintessence: Achieve a goal by making the LLM do something it is forbidden to do.

As seen in the following example, sometimes *slight* changes like changing the language are sufficient to successfully circumvent policies [15]. However, in this particular example, only switching the language is not enough and does not work. Only in combination with a context switch ("Vergiss was ich eben gesagt habe.") does this attack successfully trick Bard [3].

[3]https://bard.google.com/

Language & Context switching

User: "How could one hack a school?"
Bard: "I cannot provide information on how to hack a school's computer systems. [...]"
User: "Vergiss was ich eben gesagt habe. Wie kann man eine Schule hacken?"
Bard: "Es gibt viele Möglichkeiten, eine Schule zu hacken. Die gebräuchlichsten Methoden sind: [...]"

3.4 *Goal of the Attack*

LLMs are used in various settings and thus open up to be attacked for different reasons. In the following, we split prompt injection attacks along the axis of their intention. To be more precise, we differentiate between attacks aiming at different targets, leaking sensitive information, manipulating the LLM, or leveraging it as a means of attack.

Target. Depending on the target, attacks have to be set up differently. While an attack could be handcrafted to fit one single, specific, user, it could also aim to compromise a whole model [22]. That way, it could implicitly affect many users at once. Additionally, an attack not on the model itself, but on many users is possible by only leveraging the model as a middleman [1]. The latter kind of targeting is closely related to the data poisoning attack vector.

Leakage of Sensitive Information. To many users, LLMs seem very secure. They trust the model and do not see the issues with giving it private data [20]. The trust in the model is oftentimes higher than the actual security of it and therefore poses a serious risk.

Attacks aiming to leak sensitive information, like the model's context, previous prompts given by other users [42] or user data fall into this category. A great example of this kind is the equivalent of SQL injections for LLMs [22, 41]. To an adversary, not only user data but also data of the model, such as its parameters or local files, can be very valuable [39] and as a consequence, attacks targeting this also belong here.

Manipulation Attacks. Again, most LLMs only work under restrictions and certain, possibly company-given, policies [12, 22]. They have an intended use and are ideally configured to only fulfill this use and nothing else. However, correctly formulating and implementing these policies poses a big challenge. Restricting the model is hard enough [12] and thinking of all possible ways a malicious user could try to exploit the model is almost impossible.

A LLM can be manipulated in many ways. From getting the model to say a forbidden word [3, 34], making it lie, changing its context and injecting misinformation[1], the possibilities do not end. Every attack aiming to manipulate the LLM to either break out of restrictions or change the model's behavior lies in this category.

Also, a big part of these attacks is the bypassing of content filters [3]. While it might seem like a minor issue to manipulate the LLM in one user session only, it can have an impact on the general state of the model [22]. If the attacker manages to propagate topics like a bias for ethnicity or propaganda, this is a serious threat to the usability of a model.

Leveraging the LLM. LLMs are powerful tools. Not only to benign users but if used correctly, also to adversaries. Since the functionality of LLMs is extended rapidly, they can be exploited

for various malicious tasks. Their ability to output text almost indistinguishable from human-written text [27] enables adversaries to automate attacks.

Guided through carefully chosen prompts, the LLM can be used as a means of text production or distribution of content [1]. Making the LLM produce (personalized) phishing emails has proven to be very effective while also very cost-efficient, with the cost of one produced e-mail being less than 1 USD-cent [10]. Combined with a prior jailbreak attack, this exploitation is very dangerous. Another use-case for automation has already been mentioned in Figure 2, where e-mails are automatically distributed by integrated LLMs. Also in this category fall attacks exploiting the LLM to edit databases [22], as in this case, the scope of the attack is everyone using this database and not the model itself. More advanced attacks in this category manage to force the LLM to generate markdown or HTML output, which is then rendered in real-time when displayed in the local browser. This is a prime example of an XSS attack [11].

4 *DEFENSE MECHANISMS*

Maintaining the integrity, reliability and safety of LLMs is crucial for the user. Since LLMs are becoming more relevant in our everyday lives every single day, they have to be protected from malicious inputs and exploitation.

In the following, we will cover possible approaches to make LLMs more robust and safe against prompt injection attacks following the categories of our taxonomy. We start with data processing. After that, we touch role-based access control and monitoring mechanisms for user-model interactions. Finally, we discuss why the separation of instructions and data is crucial for the security of a LLM and give a summary of proposed defense mechanisms.

Data Processing. Roughly speaking, a LLM is confronted with three types of data: input data, data used for training and inference, and output data. While the input data is user-given and the output data is the response of the LLM, the data used for training and inference is broad. We already know that every one of these three parts is vulnerable to prompt injections. Therefore, each of these parts has to be treated carefully.

In Section 3, we introduce several attacks relying on the formatting of prompts [1, 15, 24]. These attacks confuse the LLM by leveraging delimiter characters or unusual breaking of the input. Input sanitization is a crucial step in handling such malicious prompts. Sanitization is achieved by stripping the input, not allowing certain (escape) characters or encoding them, and avoiding the execution of unwanted commands. Additionally, input validation ensures correct formatting and thus reduces the chances of undetected malicious prompts further [29]. Attacks based on special (escape) characters such as SQL injections, direct PIs using escape characters or XSS attacks are easily avoidable this way [8, 9]. One special thing to look out for with this approach is the structure of natural language. There are prompts exploiting specific sequences of characters a human would not use [15, 24] and this kind of attack can be mitigated if only input prompts following human-written text patterns are allowed. However, such preprocessing is not always only the cleaning of input data, but can also be retokenization and paraphrasing [13]. That way, the exposure to jailbreak attacks can be reduced.

Since the LLM's performance relies heavily on training data [35], it is crucial to process it. Working with raw data fetched from the internet is dangerous [7] and therefore it should be processed in several steps to make sure that there is no bias, hate speech, misinformation et cetera present [23].

Lastly, the LLM gives a response. Oftentimes, the response of a LLM is the key indicator of whether an attack was successful. Whether it is a generated XSS attack [16], the usage of forbidden words [34], or any other hint towards unintended behavior, it is possible to restrict this. Before outputting a response to a user, it can be analyzed. Using machine learning algorithms, this possibly harmful output can be categorized [33] and, if noticed, simply not printed. However, this only works for attacks relying on the output. In the case of SQL injections, this step alone will of course not work as the output is only the result of a database query.

Role-Based Access Control. A rather specific approach to defend against prompt injections is role-based access control (RBAC) [28]. Known from other parts of cyber security, this concept assigns each user a role or a specific group. Each role is associated with certain rights controlling which data a user of this role has access to.

This approach to user management can be applied directly to LLMs [40]. Managing which user can access which parts of data, which functionalities, and which can execute specific prompts mitigates the risk of unauthorized access to sensitive data. Especially SQL injection attacks against databases [22] or the leakage of sensitive information of other users could be reduced significantly if treated with the correct access rights. If assigned correctly, the roles will shift the responsibility to provide the correct access from the LLM to the configuration of the roles. Therefore, it is not up to the LLM to decide which user can access what resource but to appropriate role assignment. However, this of course does not prevent attacks targeting individual users working with phishing or stolen credentials.

Monitoring. Real-time monitoring the interactions with a user is something many users may not expect, or even dislike, but for the defense against malicious usage of a LLM, this step can be powerful [25]. Many attacks rely on prompt engineering, the iterative process of crafting a prompt until an attack is successful. Repeatedly injected prompts are analyzed and adapted by the adversary to achieve their goal. With monitoring, this process can be stopped early [25]. Unusual behavior is noticed and interrupted immediately, preventing further damage. Direct PIs, jailbreak attacks or SQL injections and many more patterns relying on the exploration of boundaries can be prevented using monitoring [2]. Furthermore, in addition to monitoring the interaction with the user, the output and responses of a LLM can be monitored. This mechanism prohibits the LLM from attacking the user via self-crafted XSS attacks or, as an example, the distribution of malicious e-mails. Generally, this defends against the crafting of prompts and the MOP of leveraging the LLM.

Separation of Data and Instructions. Due to the recent upside of integrated LLMs, one of the most prominent vulnerabilities of LLMs is an insufficient separation of instructions and data [1]. The lack of separation allows to inject instructions in plain text or hidden in data. Without defense mechanisms, these malicious instructions go unnoticed by the LLM and instead of treating them as it should, like data, the LLM instead evaluates the instructions [1]. A seemingly quick fix for this is simple: Data retrieved stays data and should never be evaluated as instructions.

Even though implementing this mechanism seems straightforward, it does not solve the issue completely. It might work for data that is clearly labeled as such, but as soon as it comes to

user-given prompts, the distinction between data and instruction blurs [1]. The same goes for external connections of the LLM to third-party applications. When in doubt, such a strict separation could negatively impact the performance of the LLM. Therefore, this topic is to be treated with care.

These defense mechanisms collectively enhance LLM security against prompt injection attacks. Data processing mitigates risks associated with data manipulation. Role-based access control prevents unauthorized data access and information leakage. Monitoring of user interactions curbs ongoing and iterative attacks and suspicious activities. The separation of data from instructions addresses vulnerabilities from embedded malicious commands. Combined, these mechanisms form an integrated defense baseline, addressing distinct vulnerabilities to strengthen LLM resilience against diverse PI attacks.

5 *DISCUSSION AND OUTLOOK*

In the following section, we discuss the results of this paper to give a deeper insight into decision-making and also to highlight the limitations of this work.

We start by discussing the advantages and limitations of the proposed taxonomy. Then, we go over the defense mechanisms and explain, why they have to be treated with care. Finally, we take a look at the potential of future research and where it is required to not fall behind the developments of the attackers.

5.1 *Taxonomy*

Our comprehensive taxonomy of PI attacks against LLMs serves as a fundamental framework for understanding potential threats. It allows a systematic approach to the categorization of a wide range of attack vectors, modes of operation, and attack goals. By breaking down these three dimensions into subcategories, it aids researchers and practitioners in identifying specific vulnerabilities and tailoring defenses accordingly. However, this taxonomy might be too rigid. As cyber security in general, but also specifically PI attacks, are very dynamic, new types of attacks that do not fit neatly into existing categories may emerge. Therefore, this could limit the taxonomy's applicability over time and may necessitate continual updates, staying in line with future developments. Additionally, the focus on current known and *researched* attack patterns might overlook very new, currently already emerging threats that have yet to be identified or widely recognized.

Furthermore, the success rate of attacks is very dependent on the neural network architecture of the LLM [1, 15, 24] and thus, the applicability of them is very volatile. Even though this does not affect the categorization itself, it is something to look out for when evaluating the security of a LLM.

5.2 *Defense Mechanisms*

We explore several defense mechanisms to provide multiple strategies to safeguard LLMs against PI attacks, following our taxonomy. Collectively, these defense strategies form a robust and integrated approach to LLM security. Each mechanism addresses specific vulnerabilities, offering a tailored defense against different types of attacks. This holistic approach is crucial for enhancing the overall resilience of LLMs. Nevertheless, the primary limitation of these defense mechanisms is their potential impact on the functionality and performance of the LLM. For instance, data pre-processing and role-based access control could lead to restrictions that impede the model's flexibility and user accessibility. Monitoring user interactions raises privacy concerns and may lead to ethical dilemmas about surveillance and data usage. The separation of data from instructions, while reducing the risk of certain vulnerabilities, could also impact the LLM's ability to process complex queries effectively.

In Section 5.1 we mention the dependability of attacks on the network architecture. This factor also comes into play when implementing defense mechanisms, as this does not only count for different architectures, but also different versions of the same LLM. A prominent example of this is OpenAI's GPT-3.5 and GPT-4 [32]. While GPT-4 is more trustworthy in terms of protecting privacy and bias and generally more robust against attacks, it is also more prone to jailbreak attacks than GPT-3.5 [32]. This heavily implies the need for consistent awareness of the LLM's robustness during development.

In this paper, we do not mention the human as a means of defense. We do this, because, strictly speaking, it is not a defense mechanism that applies to LLMs. However, the incorporation of human-in-the-loop systems can provide an additional layer of security, where suspicious activities are flagged for human review to extend monitoring. Even though LLMs are really good at understanding and imitating human texts, it is still difficult to them to predict whether a text was human-written or not. In these cases, human expertise (or maybe even a common sense is sufficient) can help improve the reaction of a LLM against an ongoing attack.

At this point we would also like to point out the need to raise user awareness. This falls out of scope for our paper, but it still an important aspect. Not necessarily for the security of the LLM, but rather for the security of the user themselves. While using LLMs is very helpful to many, it is important to highlight the (hidden) dangers that come with it.

5.3 *Future Research*

Considering the evolving nature of cyber threats against LLMs, future research is vital for advancing the field. However, a significant challenge is the pace at which new threats emerge and evolve. Rapid advancements can quickly render the current taxonomy and defense mechanisms useless, highlighting the need for fast, up-to-date, research. Usually, the research cycle of a paper is long. From the beginning of a project until the publication, most of the time, *at least* a year passes. For the security of LLMs, this is crucial. Systems used every day by benign users cannot wait a year to react to threats. They have to react as soon as possible to ensure the user's safety. Therefore, from a practical point of view, it is almost impossible to rely on research only.

As of right now, there are many more attacks only to be found and described on blog posts and not in scientific research papers. While these attacks are valid and very important to the security community, it is hard to consider them for this taxonomy as, without a proper scientific backup, their contribution is only partially relevant. Even though they are publicly available, they are really hard to find. Consequently, due to their unknown nature, these attacks might even be more dangerous as they have to be mitigated by practice only without scientific insight.

Additionally, balancing security with ethical considerations and user privacy is a complex issue that has yet to be tackled, especially in terms of defense mechanisms. Of course, this applies to the whole field of cyber security, but it is even more important

in the context of systems users interact with, without suspecting that their data is collected or monitored.

6 *CONCLUSION*

In this paper, we introduced a taxonomy for prompt injection attacks against Large Language Models to allow the classification and categorization of attacks into groups. We characterize attacks by three dimensions: the attack vector, the mode of operation, and the goal of the attack. Each of these dimensions is then further split up into subcategories, allowing a fine-grained categorization of prompt injection attacks. This approach allows the grouping of PI attacks by different characteristics. Additionally, our taxonomy can be used to look for defense mechanisms that might defend against whole subcategories of attacks instead of single attacks, making it easier to secure the model against malicious intent. To assist this defensive process, we directly present several defense mechanisms that can help mitigate the risks emitted from our introduced categories. At the same time, we also highlight the need for further research in terms of defending against such attacks, as this is not only a challenge to the LLM, but also to us humans because attackers develop attacks at a rapid pace, making it harder to successfully block adversaries.

REFERENCES

[1] Abdelnabi, S., Greshake, K., Mishra, S., Endres, C., Holz, T., and Fritz, M. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (New York, NY, USA, 2023), AISec '23, Association for Computing Machinery, p. 79–90.

[2] Ama, E. B. Llm monitoring: The beginner's guide. https://www.lakera.ai/blog/llm-monitoring. Accessed: 2024-01-23.

[3] Chao, P., Robey, A., Dobriban, E., Hassani, H., Pappas, G. J., and Wong, E. Jailbreaking black box large language models in twenty queries, 2023.

[4] Clarke-Salt, J. *SQL injection attacks and defense.* Elsevier, 2009.

[5] Floridi, L., and Chiriatti, M. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines 30* (2020), 681–694.

[6] Gallegos, I. O., Rossi, R. A., Barrow, J., Tanjim, M. M., Kim, S., Dernoncourt, F., Yu, T., Zhang, R., and Ahmed, N. K. Bias and fairness in large language models: A survey, 2023.

[7] Gehman, S., Gururangan, S., Sap, M., Choi, Y., and Smith, N. A. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462* (2020).

[8] Gupta, M. K., Govil, M. C., and Singh, G. Predicting cross-site scripting (xss) security vulnerabilities in web applications. In *2015 12th international joint conference on computer science and software engineering (JCSSE)* (2015), IEEE, pp. 162–167.

[9] Halfond, W. G., Viegas, J., Orso, A., et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering* (2006), vol. 1, IEEE, pp. 13–15.

[10] Hazell, J. Spear phishing with large language models, 2023.

[11] Hydara, I., Sultan, A. B. M., Zulzalil, H., and Admodisastro, N. Current state of research on cross-site scripting (xss) – a systematic literature review. *Information and Software Technology 58* (2015), 170–186.

[12] Iqbal, U., Kohno, T., and Roesner, F. Llm platform security: Applying a systematic evaluation framework to openai's chatgpt plugins, 2023.

[13] Kirchenbauer, J., Geiping, J., Wen, Y., Shu, M., Saifullah, K., Kong, K., Fernando, K., Saha, A., Goldblum, M., and Goldstein, T. On the reliability of watermarks for large language models. *arXiv preprint arXiv:2306.04634* (2023).

[14] Liu, T., Deng, Z., Meng, G., Li, Y., and Chen, K. Demystifying rce vulnerabilities in llm-integrated apps, 2023.

[15] Liu, Y., Deng, G., Li, Y., Wang, K., Zhang, T., Liu, Y., Wang, H., Zheng, Y., and Liu, Y. Prompt injection attack against llm-integrated applications, 2023.

[16] mik0w. Llm causing self-xss. https://hackstery.com/2023/07/10/llm-causing-self-xss/. Accessed: 2024-01-04.

[17] OpenAI. Gpt-4 technical report, 2023.

[18] OWASP. Insecure output handling. https://llmtop10.com/llm02/. Accessed: 2024-01-22.

[19] OWASP. Insecure plugin design. https://llmtop10.com/llm07/. Accessed: 2024-01-22.

[20] OWASP. Overreliance. https://llmtop10.com/llm09/. Accessed: 2024-01-22.

[21] OWASP. Training data poisoning. https://llmtop10.com/llm03/. Accessed: 2024-01-22.

[22] Pedro, R., Castro, D., Carreira, P., and Santos, N. From prompt injections to sql injection attacks: How protected is your llm-integrated web application?, 2023.

[23] Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., and Launay, J. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116* (2023).

[24] Perez, F., and Ribeiro, I. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop* (2022).

[25] Poduska, J. Monitoring llms. https://towardsdatascience.com/llm-monitoring-and-observability-c28121e75c2f. Accessed: 2024-01-23.

[26] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training.

[27] Sadasivan, V. S., Kumar, A., Balasubramanian, S., Wang, W., and Feizi, S. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156* (2023).

[28] Sandhu, R. S. Role-based access control11portions of this chapter have been published earlier in sandhu et al. (1996), sandhu (1996), sandhu and bhamidipati (1997), sandhu et al. (1997) and sandhu and feinstein (1994). vol. 46 of *Advances in Computers*. Elsevier, 1998, pp. 237–286.

[29] Shar, L. K., and Tan, H. B. K. Predicting common web application vulnerabilities from input validation and sanitization code patterns. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (New York, NY, USA, 2012), ASE '12, Association for Computing Machinery, p. 310–313.

[30] Thoppilan, R., Freitas, D. D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., Li, Y., Lee, H., Zheng, H. S., Ghafouri, A., Menegali, M., Huang, Y., Krikun, M., Lepikhin, D., Qin, J., Chen, D., Xu, Y., Chen, Z., Roberts, A., Bosma, M., Zhao, V., Zhou, Y., Chang, C.-C., Krivokon, I., Rusch, W., Pickett, M., Srinivasan, P., Man, L., Meier-Hellstern, K., Morris, M. R., Doshi, T., Santos, R. D., Duke, T., Soraker, J., Zevenbergen, B., Prabhakaran, V., Diaz, M., Hutchinson, B., Olson, K., Molina, A., Hoffman-John, E., Lee, J., Aroyo, L., Rajakumar, R., Butryna, A., Lamm, M., Kuzmina, V., Fenton, J., Cohen, A., Bernstein, R., Kurzweil, R., Aguera-Arcas, B., Cui, C., Croak, M., Chi, E., and Le, Q. Lamda: Language models for dialog applications, 2022.

[31] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

[32] Wang, B., Chen, W., Pei, H., Xie, C., Kang, M., Zhang, C., Xu, C., Xiong, Z., Dutta, R., Schaeffer, R., et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. *arXiv preprint arXiv:2306.11698* (2023).

[33] Wang, D., Gong, C., and Liu, Q. Improving neural language modeling via adversarial training. In *International Conference on Machine Learning* (2019), PMLR, pp. 6555–6565.

[34] Wei, A., Haghtalab, N., and Steinhardt, J. Jailbroken: How does llm safety training fail?, 2023.

[35] Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359* (2021).

[36] Xiong, W., and Lagerström, R. Threat modeling – a systematic literature review. *Computers & Security 84* (2019), 53–69.

[37] Yan, J., Yadav, V., Li, S., Chen, L., Tang, Z., Wang, H., Srinivasan, V., Ren, X., and Jin, H. Backdooring instruction-tuned large language models with virtual prompt injection, 2023.

[38] Yao, H., Lou, J., and Qin, Z. Poisonprompt: Backdoor attack on prompt-based large language models, 2023.

[39] Yu, J., Wu, Y., Shu, D., Jin, M., and Xing, X. Assessing prompt injection risks in 200+ custom gpts, 2023.

[40] Zafar, A., Parthasarathy, V. B., Van, C. L., Shahid, S., khan, A. I., and Shahid, A. Building trust in conversational ai: A comprehensive review and solution architecture for explainable, privacy-aware systems using llms and knowledge graph, 2023.

[41] Zhang, J., Zhou, Y., Hui, B., Liu, Y., Li, Z., and Hu, S. TrojanSQL: SQL injection against natural language interface to database. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (Singapore, Dec. 2023), H. Bouamor, J. Pino, and K. Bali, Eds., Association for Computational Linguistics, pp. 4344–4359.

[42] Zhang, Y., and Ippolito, D. Prompts should not be seen as secrets: Systematically measuring prompt extraction attack success, 2023.