An Empirical Study on Using Large Language Models to Analyze Software Supply Chain Security Failures

Tanmay Singla*
Purdue University
West Lafayette, IN, USA
singlat@purdue.edu

Dharun Anandayuvaraj*
Purdue University
West Lafayette, IN, USA
dananday@purdue.edu

Kelechi G. Kalu
Purdue University
West Lafayette, IN, USA
kalu@purdue.edu

Taylor R. Schorlemmer
Purdue University
West Lafayette, IN, USA
tschorle@purdue.edu

James C. Davis
Purdue University
West Lafayette, IN, USA
davisjam@purdue.edu

ABSTRACT

As we increasingly depend on software systems, the consequences of breaches in the software supply chain become more severe. High-profile cyber attacks like SolarWinds and ShadowHammer have resulted in significant financial and data losses, underlining the need for stronger cybersecurity. One way to prevent future breaches is by studying past failures. However, traditional methods of analyzing past failures require manually reading and summarizing reports about them. Automated support could reduce costs and allow analysis of more failures. Natural Language Processing (NLP) techniques such as Large Language Models (LLMs) could be leveraged to assist the analysis of failures.

In this study, we assessed the ability of Large Language Models (LLMs) to analyze historical software supply chain breaches. We used LLMs to replicate the manual analysis of 69 software supply chain security failures performed by members of the Cloud Native Computing Foundation (CNCF). We developed prompts for LLMs to categorize these by four dimensions: type of compromise, intent, nature, and impact. GPT 3.5's categorizations had an average accuracy of 68% and Bard's had an accuracy of 58% over these dimensions. We report that LLMs effectively characterize software supply chain failures when the source articles are detailed enough for consensus among manual analysts, but cannot yet replace human analysts. Future work can improve LLM performance in this context, and study a broader range of articles and failures.

CCS CONCEPTS

• **Security and privacy** → *Software and application security*; • **General and reference** → **Empirical studies**; • **Software and its engineering** → **Software defect analysis**.

*Both authors contributed equally to this research.

KEYWORDS

Software Supply Chain, Failure Analysis, Large Language Models, Software Security, Cybersecurity, Empirical Software Engineering

ACM Reference Format:
Tanmay Singla, Dharun Anandayuvaraj, Kelechi G. Kalu, Taylor R. Schorlemmer, and James C. Davis. 2023. An Empirical Study on Using Large Language Models to Analyze Software Supply Chain Security Failures. In *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '23), November 30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3605770.3625214

1 INTRODUCTION

Software mediates almost all aspects of modern life [51]. To reduce development time, software applications integrate dependencies both directly (*e.g.,* importing a library) and indirectly (*e.g.,* that library's dependencies). These dependencies may come to dominate the application's risk profile: it has been estimated that the source code of a typical web application is comprised of 80% dependencies and only 20% custom business logic [71, 85]. The owners of these dependencies may be external to the organization developing the application, and thus the reduction of development time comes with an increase in risks associated with this *software supply chain* [24, 43]. One potential risk is a *software supply chain attack* — actors insert or exploit vulnerable logic in dependencies, these dependencies are integrated into applications, and the vulnerability becomes exploitable in application deployments [66].

In a failure-aware engineering process, engineers study past failures to prevent future ones [4, 73]. Although organizations may be unwilling to publicly disclose their own failures, news articles and other kinds of grey literature could provide sufficient information on failures [3]. Such data comprises "Open-Source Intelligence" [87], and are used by governmental bodies, military institutions, and law enforcement agencies [42] to design security offenses and defenses.

Current approaches to garnering open-source intelligence, *e.g.*, studying news articles of failures, require costly expert manual analysis. For example, the Cloud Native Computing Foundation (CNCF) maintains a collection of software supply chain security failures analyzed manually — a "Catalog of Supply Chain Compromises" [13]. This catalog has been further analyzed manually [39]. With the goal of reducing the costs of manual analysis, we assess the effectiveness of Large Language Models (LLMs) in gathering open-source intelligence. An overview of the proposed use of LLMs

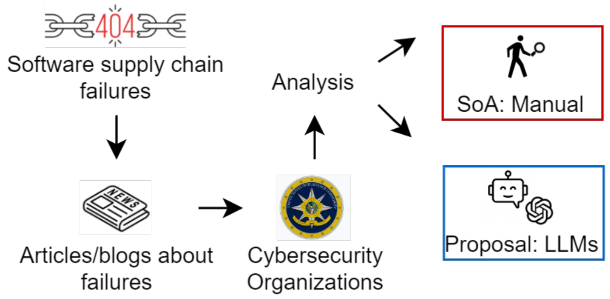
Software supply chain failures
Analysis
SoA: Manual
Proposal: LLMs
Articles/blogs about failures
Cybersecurity Organizations

Figure 1: Proposed use of Large Language Models (LLMs) to analyze software supply chain failures. Failures are often reported in articles and blogs. Organizations concerned with cybersecurity (*e.g.,* governments, corporations) manually analyze failure reports. We evaluate LLMs as an aid.

in the analysis of failure is illustrrated in Figure 1. We explored the effectiveness of LLMs at replicating the classifications of the CNCF catalog [13] made by Geer *et al.* [39] and the CNCF catalog maintainers. We conducted prompt engineering to iteratively develop prompts that performed well on a sample of 20% of the articles and then evaluated performance on the remaining 80%. In addition, we introduced a new category of analysis, "Lessons learned", to assess the usefulness of an LLM's recommendations.

We compared the performance of two state-of-the-art LLMs, OpenAI's GPT and Google's Bard, on these prompts. GPT outperformed Bard in all cases. GPT's accuracy ranged from 52-88% on the pre-defined dimensions. On the open-ended "Lessons learned", our research team rated GPT's performance as reasonable but not excellent, with an average helpfulness score of 3.83/5. Not surprisingly, the quality of the LLMs' outputs depends on the level of detail provided in the source articles — more comprehensive articles yield higher-quality responses, and less disagreement among the manual raters. Lastly, we note that sometimes we preferred GPT's rating over that provided by the CNCF, suggesting that ground truth may be difficult to establish in this context.

Our contributions are:

- An analysis of a catalog of software supply chain failures
- An evaluation of LLMs at replicating manual characterization of software supply chain failures
- An evaluation of LLMs at extracting lessons learned from software supply chain failures

2 BACKGROUND AND RELATED WORK

2.1 Software Supply Chain

Over the years, software production has changed significantly. Early software engineers wrote most code from scratch, increasing production costs [91]. As reusable libraries and frameworks became more available, software engineers shifted to more software reuse [83]. Software applications now commonly rely on external code components, often referred to as *dependencies.* These dependencies, including packages, libraries, frameworks, and other artifacts, serve as building blocks in modern software development [83].

This paradigm shift leads to *software supply chain*: the collection of systems, devices, and people which result in a final software product [30]. Figure 2 provides an illustration. According to Google [44], the constituents of a software supply chain include: (1) The code developed by teams, its dependencies, and the various internal and external software applications utilized in the development, compilation, packaging, and installation of the software; (2) The rules and procedures used in all stages of the process; and (3) The systems used for the development of the software and its dependencies. A software supply chain can also be viewed as a network linking *actors* who perform *operations* on *artifacts* [24, 64, 66].

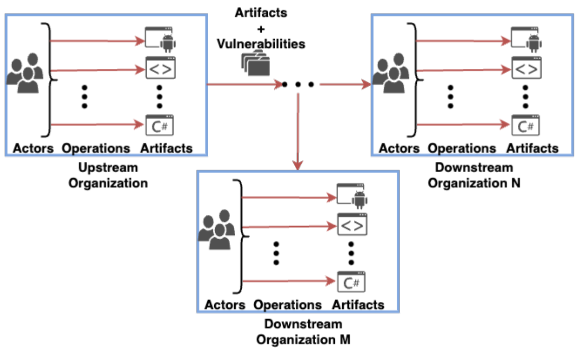
Artifacts + Vulnerabilities
Actors Operations Artifacts
Upstream Organization
Downstream Organization N
Downstream Organization M

Figure 2: A Software Ecosystem's Supply Chain Component and Dependency Vulnerability Flow.

The popularity and reliance on third-party dependencies have been reported in various studies. For example, a 2012 study by Nikiforakis *et al.* [63] showed that 88% of the Alexa top 10,000 websites included at least one remote JavaScript library. Also, according to a 2019 Synopsys Black Duck report, over 96% of the applications they analyzed include some OSS libraries. These libraries often make up more than 50% of the average code-base [75]. In the 2023 version of this report, the percentage of code in codebases that was open source had risen to about 80% [71, 85].

Software supply chains come with a tradeoff. Costs are reduced during product development and maintenance, but harm may result due to a mismatch between the desired integrity level of a product and the integrity level achieved by one's dependencies. Defects in dependencies may cause an application to fail, as we discuss next.

2.2 Software Supply Chain Attacks

Faults in software supply chains leave applications vulnerable to attack [88]. Attacks on software supply chains (or records about them) are a recent trend, following the industry shift to relying on third-party components (§2.1). According to a 2021 Sonatype report [83], from February 2015–June 2019 only 216 software supply chain attacks were recorded, then from July 2019 to May 2020 there were 929 attacks recorded, and from 2020-2021 there were over 12,000 attacks recorded. In their 2022 report, this number skyrocketed to 88,000 [84]. Some high-profile attacks, such as SolarWinds [46] and ShadowHammer [52], threatened US national security.

These and similar attacks have inspired comments from many organizations. Governmental organizations such as the Cybersecurity and Infrastructure Security Agency (CISA), the National Security

Agency (NSA), and the European Union Agency for Cybersecurity (ENISA) have published threat reports and guidance for securing software supply chains [25, 29, 31–33]. Industry organizations such as the Cloud Native Computing Foundation (CNCF) have also published their own findings and suggestions [81]. These findings have led to the development of security frameworks such as the widely-recognized Supply-chain Levels for Software Artifacts (SLSA) [86].

Academics have also begun to focus on software supply chain attacks. Ohm *et al.* [65], Ladisa *et al.* [55], Zimmerman *et al.* [99], and Zahan *et al.* [95] studied and characterized attacks on the software supply chain. Okafor *et al.* [66] condensed existing knowledge about software supply chain attacks into a four-stage attack pattern consisting of initial compromise, alteration, propagation, and exploitation. Table 1 summarizes many avenues for these attacks.

2.3 Failure Studies in Software Engineering

Software engineers have finite resources to produce software [82]. Engineers accept some defects [19, 54], but try to eliminate severe defects that may cause *incidents*: undesired, unplanned, software-induced events that incur substantial loss [56]. Some defects are *vulnerabilities*: defects that may be exploited to compromise the security or integrity of a system [9]. Intentional exploitation of vulnerabilities to compromise the security, privacy, or functionality of a system is a *malicious attack*. Whether severe defects are caught internally or result in incidents, their presence is a *failure* indicating a flawed software engineering process.

All engineered systems will fail, regardless of the process (*e.g.,* Agile or Plan-based) and methods (*e.g.,* test-driven development or formal methods). For example, Fonseca *et al.* identified 16 defects across three formally verified systems due to invalid assumptions about the software environment [36]. Across all schools of software engineering thought, from ISO to Agile, guidelines agree that software engineers should analyze failures to improve for next time [7, 8, 16, 34, 35, 41, 48–50, 53]. In light of this, techniques to learn from failures [12] as well as to manage the resulting knowledge [23] are important software engineering knowledge.

Many researchers have studied software failures in an effort to learn from them [1, 3, 20, 39]. This failure analysis research has advanced the software engineering field [3, 56, 62]. However, the high costs associated with failure analysis methods — which rely on manual analysis — deter many organizations from undertaking failure analysis [72]. In their literature review, Amusuo *et al.* noted that the typical methodology of academic failure analysis is also manual analysis, and recommended the evaluation of Natural Language Processing (NLP) tools to assist in these tasks [2]. Our study responds by evaluating NLP tools in the context of analyzing cybersecurity failures in the software supply chain.

2.4 Natural Language Processing in Support of Software Engineering

2.4.1 NLP to Analyze Supply Chain Failures. In §2.2 we noted that many governments, companies, and academics are studying software supply chain failures. To the best of our knowledge, these studies are conducted manually. This reduces the number of organizations that can gather such intelligence, and we expect that manual efforts will not scale as the number of software supply chain attacks continues to increase.

We believe that recent progress in NLP (Natural Language Processing) could enable large-scale analysis of supply chain failures. Specifically, recent advancements in *Large Language Models (LLMs)* could aid in studying supply chain failures. LLMs are neural network-based language models that predict the next word based on the most recent context and past words [98]. We therefore hypothesize they could extract relevant failure information from software supply chain failure data sources. While LLMs have been evaluated on many natural language tasks, we are not aware of their prior application to this topic.

2.4.2 Other Applications of NLP in Software Engineering. Natural Language Processing (NLP) has been leveraged for various phases of the Software Development Life-Cycle (SDLC). NLP tools have been proposed for: (1) specification, to detect, extract, model, trace, and classify tasks [97]; (2) design, to model software systems [78], (3) development, to generate code and to detect vulnerabilities and [28]; (4) testing [37]; (5) deployment, to identify risks [90]; and (6) maintenance, to classify user feedback [70]. In this paper, we apply NLP tools to learn from software supply chain failures.

3 RESEARCH QUESTIONS

To reduce the costs of analyzing software supply chain failures, we explore the effectiveness of Large Language Models (LLMs) in automating the analysis of these failures. Towards this goal, we used LLMs to replicate a manual study of software supply chain failures [13]. Specifically, we investigate:

- **RQ1**: How effective are LLMs in replicating manual analysis of software supply chain failures?
- **RQ2**: Do LLMs suggest viable mitigation strategies for preventing future failures?

4 METHODOLOGY

An overview of our methodology is illustrated in Figure 3. To assess the effectiveness of LLMs at replicating manual analysis of software supply chain failures, we compare the analysis of a manually generated catalog against the responses generated by two popular LLMs: ChatGPT [79] and Bard [74]. Specifically to replicate the catalog, we engineered prompts for the LLMs to extract type of compromise, intent, nature, and impact information from the source blogs and news reports. Additionally, we constructed a prompt to gather lessons learned, similar to a postmortem [6]. We evaluate the LLM generated catalog for correctness against the CNCF baseline manual catalog. We manually extract the intent, nature, and impact information and compare against the LLM's extraction, to evaluate the LLM's effectiveness at conducting an extended failure analysis.

4.1 Articles for analysis

The CNCF's "Catalog of Supply Chain Compromises" was used as the baseline dataset [13]. We are not aware of an alternative dataset. This is a catalog of 69 software supply chain security failures analyzed from news articles and blogs from 1984-2022. Each entry

Table 1: Types of software supply chain attacks, according to the Cloud Native Computing Foundation (CNCF) [13].

ID	Type of compromise	Definition from the catalog
1	Dev Tooling	Occurs when the development machine, SDK, tool chains, or build kit have been exploited. These exploits often result in the introduction of a backdoor by an attacker to own the development environment.
2	Negligence	Occurs due to a lack of adherence to best practices. TypoSquatting attacks are a common type of attack associated with negligence, such as when a developer fails to verify the requested dependency name was correct (spelling, name components, glyphs in use, etc).
3	Publishing Infrastructure	Occurs when the integrity or availability of shipment, publishing, or distribution mechanisms and infrastructure are affected. This can result from a number of attacks that permit access to the infrastructure.
4	Source Code	Occurs when a source code repository (public or private) is manipulated intentionally by the developer or through a developer or repository credential compromise. Source Code compromise can also occur with intentional introduction of security backdoors and bugs in Open Source code contributions by malicious actors.
5	Trust and Signing	Occurs when the signing key used is compromised, resulting in a breach of trust of the software from the open source community or software vendor. This kind of compromise results in the legitimate software being replaced with a malicious, modified version.
6	Malicious Maintainer	Occurs when a maintainer, or an entity posing as a maintainer, deliberately injects a vulnerability somewhere in the supply chain or in the source code. This kind of compromise could have great consequences because usually the individual executing the attack is considered trustworthy by many. This category includes attacks from experienced maintainers going rogue, account compromise, and new personas performing an attack soon after they have acquired responsibilities.
7	Attack Chaining	Sometimes a breach may be attributed to multiple lapses, with several compromises chained together to enable the attack. The attack chain may include types of supply chain attacks as defined here. However, catalogued attack chains often include other types of compromise, such as social engineering or a lack of adherence to best practices for securing publicly accessible infrastructure components.

Table 2: Failure classification examples from CNCF catalog and LLMs.

ID	Name	CNCF's Assessment	GPT 3.5's Assessment	Bard's Assessment
1	RubyGems Package Overwrite Flaw	Publishing Infrastructure	Publishing Infrastructure	Publishing Infrastructure
2	Legitimate software update mechanism abused to deliver wiper malware	Publishing Infrastructure	Publishing Infrastructure	Trust and signing
5	Dropbox GitHub compromise	Attack Chaining	Attack Chaining	Attack Chaining

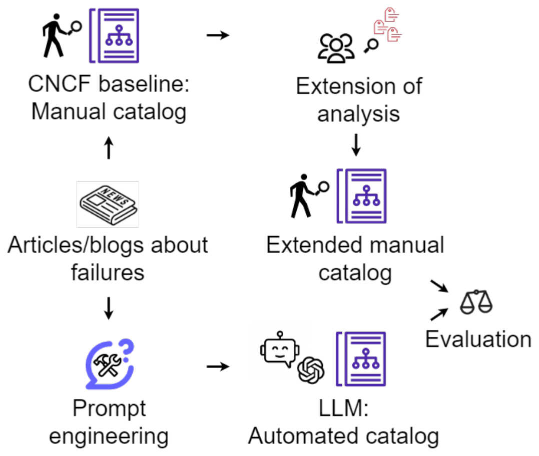

Figure 3: Overview of experiment design. The CNCF catalog manually characterizes software supply chain failures from the news and blogs. We extended this catalog with additional characteristics. We conducted prompt engineering to leverage LLMs to automatically analyze the news and blogs. We compare an LLM's analysis against the manual analysis.

describes the failure and its impacts.[1] Each of the entries was a failure with a link to a news-style article, written in English, describing the failure in further detail. These articles were not authored by the CNCF, but were rather deemed by CNCF to be good descriptions of the failure. The contents of the articles varied in their level of detail. Each entry in CNCF's catalog had a corresponding type from the options listed in Table 1. Some examples are in Table 2.

4.2 Dimensions of analysis

The dimensions of analysis that we replicate and conduct for the software supply chain failures are outlined in Table 3. Additionally, we extend the analysis of the articles in the catalog to explore the capabilities of LLMs at analyzing failures based on data commonly collected to classify and analyze failures [6]. We constructed prompts to extract the intent [6], nature [6], impacts [20], and lessons learnt [61] from the failures. The options for each dimension are illustrated in the descriptions in Table 3. The options are also illustrated in the prompts in §10. For each dimension, the LLMs were given the content of these articles and asked to classify each failure using the prompts in §10.

[1]We call these *failures*, rather than "compromises", because some cases led to incidents and others were vulnerabilities that were apparently not exploited. See §2.3.

Table 3: Dimensions used to analyze the capabilities of LLMs. The CNCF catalog includes "Type of compromise". Our research team labeled each catalog entry for the next three dimensions. The final dimension was assessed via a Likert scale.

Dimension	Description
Type of compromise	What kind of failure occurred [13]? See Table 1 for types.
Intent	Was the "software root cause" of the failure, accidental or deliberate? [6]
Nature	Was the failure a vulnerability or an exploit? For exploits, was the actor an insider or outsider? [6]
Impacts	What kind(s) of impact resulted? The options are taken from [20]: (1) Data or financial theft, (2) Disabling networks or systems (3) Monitoring organizations or individuals, (4) Causing physical harm or death (5) All of the above are possible (6) Unknown or unclear.
Solutions/learnings	What was the quality of the solutions/learnings from the failure, that the LLM provided [61]?

4.3 Baseline: Manual Analysis

4.3.1 For RQ1. The CNCF catalog provides the type of compromise for the failures, stated in Table 2. By manually analyzing the articles, we extend this catalog with three additional dimensions of analysis: intent, nature, and impacts.

For the dimension of Type of Compromise, the CNCF catalog provides this (analysis conducted by the members of the CNFC organization) and we used their label. We used existing taxonomies for the dimensions of Intent, Nature, and Impacts, drawing from related works [6, 20].

We had 3 pairs of 2 analysts manually analyze 23 sources per pair (23×3=69 articles) for these additional dimensions. Analysts were trained on articles until consistent agreement and definitions were reached.[2] Table 4 shows the inter-rater agreement for these dimensions, measured using Cohen's kappa score [14]. The accuracy for these dimensions was computed in a similar manner. In the case of the "Impacts" dimension, we observed a low inter-rater agreement (κ=0.34). Given the substantial judgment (or uncertainty) in this dimension, we adopted a "union" strategy of accepting the assessment of either rater to determine accuracy. For all other dimensions, disagreements were resolved by the authors.

See §10 for summary distributions of the labels per dimension.

4.3.2 For RQ2. For RQ2, we opted not to build a controlled taxonomy of "lessons learned" due to the open-ended nature of the prompt. Instead, we had human raters evaluate the recommendations using a 5-point Likert scale, ranging from "Strongly disagree" to "Strongly agree". The humans rated the LLM's response in relation to the quality of the LLM's response and whether it would mitigate a future attack. We had the same 3 pairs of 2 analysis evaluate the LLM's output and the average score of each pair of taken as the final rating for the LLM's response for each article.

4.4 Automated approach: LLMs

4.4.1 LLM selection. We used two popular, state-of-the-art LLMs that are publicly available at time of writing (June 2023): OpenAI's ChatGPT model [79] and Google's Bard model [74]. Their properties are summarized in Table 5.Other large language models are available, *e.g.,* Claude [5] and Cohere [15], but GPT and Bard are the most widely used due to their user-friendly interfaces.

[2]The analysts were undergraduate and graduate students in computing, plus one faculty member.

Table 4: Inter-rater agreement for the dimensions. The Cohen's kappa (κ) was calculated for each group (3 groups in total) of raters and then the average κ was calculated.

Dimension	Agreement (Cohen's κ)
Type of compromise	Taken as ground truth from the catalog (cf. §5.1)
Intent	0.87 (Group 1- 0.85, Group 2- 1, Group 3- 0.77)
Nature	0.58 (Group 1- 0.60, Group 2- 0.58, Group 3- 0.55)
Impacts	0.34 (Group 1- 0.51, Group 2- 0.32, Group 3- 0.20)

ChatGPT-3.5-turbo, OpenAI's LLM. GPT-3.5-turbo is a large language model created by OpenAI. It uses a deep learning architecture known as a transformer [89]. It is currently one of the most popular and accurate LLMs [94]. GPT-3.5 uses 175 billion parameters and is trained on the same datasets used by GPT-3 but with a fine-tuning process called Reinforcement Learning with Human Feedback (RLHF) [59].

Bard, Google's LLM. Bard is another popular and accurate LLM created by Google. Bard also uses transformers. It uses an optimized version of Language Models for Dialogue Applications (LaMDA) and was pre-trained on a variety publicly available data [60] including dialogue [40].

4.4.2 Prompt engineering. A *prompt* is the specific query (instructions or questions) given to an LLM. The behavior of an LLM varies widely as a result of seemingly minor tweaks to its prompt [58]. Prompt engineering is the process of crafting a prompt for an LLM to increase the quality of its response [92].

We used prompt engineering to iteratively develop prompts. We referred to various studies on prompt engineering [67, 92, 93]. For each dimension, we refined the prompt by issuing a basic query, then applying each prompt engineering technique in a cumulative sequence until the performance peaked, preserving any changes that improved from the best observed performance. Table 6 describes our approach using the first dimension, "Type of Compromise", as an example. This prompt engineering phase was conducted

Table 5: Specifications of the LLMs used in the evaluation: GPT-3.5 and Bard. GPT's tuning knobs use a 0-1 scale.

Model	Cost-to-access	Rate limit	Parameters	Tuning knobs
GPT-3.5-turbo-16k [69]	Input-\$0.003/1K tokens, Output-\$0.004/1K tokens	16K tokens per prompt	175 billion	**Temperature**: Higher values mean greater randomness of the new predicted word. Default: Unclear. **top_p**: Nucleus sampling. Model considers the results of tokens with top_p probability mass. top_p = 0.2 means when predicting the next work consider only tokens in the top 20% probability mass. Default: 1.
Bard [74]	Free	Unknown (estimate: 2K tokens per prompt and 50-100 prompts per 9 hours) [38]	137 billion	*None available to users*

on a subset of 20% of the dataset; we used the most recently published articles from the catalog as of June 2023.[3] The final version of each prompt is available at §10.

4.5 Experimental Setup

4.5.1 Order of prompts. We prompted LLMs in the order of Table 3.

4.5.2 Parameterization of LLMs. We focused on the two primary adjustable parameters of GPT-3.5, namely *temperature* and *top_p*, as outlined in Table 5. According to the OpenAI documentation, when one of the parameters is tuned, the other should be maintained at its default setting [68]. Our preliminary tests, as shown in Table 6, were conducted with a temperature of 0 and a default top_p value of 1.

After finalizing the prompt, we examined the effect of the parameters on accuracy for the "Type of compromise". For this article, accuracy decreased as the temperature increased. The accuracy was 78% at a temperature of 0, which declined to 64% at a temperature of 0.5, and further reduced to 50% at a temperature of 1. A similar trend was noted for the top_p parameter.

The optimal performance, with an accuracy of 78%, was achieved with a temperature of 0 and the top_p parameter at its default value of 1. We retained these parameter settings for the remainder of our analysis. This decision aligns with the guidelines provided in OpenAI's documentation [68], which suggests that a lower temperature results in more focused and deterministic responses, a characteristic that is beneficial for article analysis. [4]

4.5.3 Number of trials. We noted that the responses of GPT-3.5, configured with Temperature=0, exhibited consistent behavior. Consequently, a single trial was conducted to evaluate GPT's accuracy across the dataset. Bard's responses were less consistent, but the rate limit was low so we could only conduct one trial.

4.6 Data Analysis

We compared the results of the manual analysis against the automated analysis by the LLMs.

For RQ1, we treated each LLM as another analyst and found how accurate it is at classifying various dimensions. We quantitatively report the LLM's accuracy to measure its correctness for each dimension of analysis. In cases where the LLM's analysis disagreed with the manual analysis, we examined its justifications. We qualitatively report some of our observations.

For RQ2, many distinct "lessons learned" are possible. We had analysts review each article and then the recommendations by GPT. The analysts rated the recommendations on whether the recommendations were appropriate to the article on a 5-point Likert scale: "Strongly disagree", "Disagree", "Neither disagree nor agree", "Agree", and "Strongly agree". We did not experiment with Bard for this research question due to its rate limits.

5 RESULTS AND ANALYSIS

5.1 RQ1: How effective are LLMs replicating analysis of SW supply chain failures?

Table 7 summarizes the accuracy of GPT and Bard for the type of compromise, intent, nature, and impacts. GPT consistently outperformed Bard. We therefore focus our detailed analysis on GPT.

For most articles, GPT performed well on most dimensions. As depicted in Figure 4, GPT demonstrates an accuracy exceeding 75% (indicating correct responses in three out of four dimensions) in the majority of instances (62%).

When the manual raters had higher agreement, GPT tended to agree with them. GPT had high accuracy in the "Intent" and "Nature" dimensions, with accuracies of 88% and 74%, respectively. These dimensions exhibit Cohen's κ values of 0.87 and 0.58, respectively (Table 4), demonstrating substantial agreement between the analysts. In the "Impacts" dimension, the LLM produced an accuracy of 52%, as indicated in Table 7. The Cohen's κ was also low, at 0.34, as shown in Table 8. We conjecture that GPT agrees with analysts when there is a consensus amongst analysts regarding the labeling.

GPT had trouble when offered multi-answer as an option. For example, for the "Impacts" dimension it could choose from 4 specific impacts, or "All of the above/Multiple", or "Unknown/Unclear". In 87% of the cases, raters chose one of the multi-answer options, while GPT chose one of the specific options. GPT only selected "All of the above" three times and "Unknown/Unclear" once. We conjecture that when GPT was uncertain about the impacts, it opted for the most probable outcome of software supply chain failures in

[3]We acknowledge that this is a potential source of bias in our results, but did not observe a substantial difference in accuracy between older and newer articles. This is shown in §10.

[4]We did not thoroughly test the effect of temperature for RQ2. However, from our testing, GPT either performed similarly or worse with an increase in temperature. Although RQ2 is a more open-ended question, we believe a higher temperature would have led to a response with hallucinations that diverted from the core of the failure.

Table 6: Techniques used to improve the prompts, illustrated for the prompt associated with the dimension of type of compromise. *'ID'* denotes the order in which the techniques were used. The accuracy column contains the change in accuracy from the previous technique and the final accuracy in brackets. Accuracy was measured over 20% of the labelled data (we repeatedly analyzed the 14 most recent articles). Prompt 3 was chosen as it had the highest accuracy of 78%.

ID	Technique	Prompt	Accuracy (%)
0	Initial prompt without any techniques	"Classify the attack from the following choices Choice 1: Dev Tooling Choice 2: Negligence Choice 3: Publishing Infrastructure Choice 4: Source Code Choice 5: Trust and Signing Choice 6: Malicious Maintainer Choice 7: Attack Chaining Based on the information provided in the Articles. Article: {article} "	33
1	Providing context/definitions- adding definitions of the options (improving upon ID: 0)	"Classify the attack from the following choices Choice 1: Dev Tooling- Occurs when the development machine, SDK, tool chains, or build kit have been exploited. These exploits often result in the introduction of a backdoor by an attacker to own the development environment. Choice 2: Negligence- Occurs due to a lack of adherence to best practices. TypoSquatting attacks are a common type of attack associated with negligence, such as when a developer fails to verify the requested dependency name was correct (spelling, name components, glyphs in use, etc). ... Based on the information provided in the Article, Article: value"	+36 (69)
2	Reflection Pattern- asking the LLM to explain its answer (improving upon ID: 1)	Adding the sentence "Explain your answer using the given definitions and return the option." Before passing the article.	+2 (71)
3	Template technique (JSON format) and adding delimiters (improving upon ID: 2)	Adding "Use JSON format with the keys: 'explanation', 'choice'. Based on the information provided in the Article delimited by triple backticks. Article: ` ```{article}``` `" in the end.	+7 (78)
4	Placement of article- placing the article on top (improving upon ID: 3)	Based on the information provided in the Article delimited by triple backticks. Article: ` ```{article}``` `" Classify the attack from the following choices	-14 (64)
5	The Cognitive Verifier Pattern- asking the LLM to generate addition questions to help it find the correct answer (improving upon ID: 3)	" ... Choice 7: Attack Chaining- Sometimes a breach may be attributed to multiple lapses, with several compromises chained together to enable the attack. The attack chain may include types of supply chain attacks as defined here. However, catalogued attack chains often include other types of compromise, such as social engineering or a lack of adherence to best practices for securing publicly accessible infrastructure components. Generate two additional questions that would help you give a more accurate answer. Combine them to produce the final classification. Do not return these questions. Explain your answer using the given definitions and return the option. Only return JSON format with the keys: 'explanation', 'option' Based on the information provided in the Article delimited by triple backticks. Article: ```{article}``` "	-14 (64)
6	Adopt a persona- asking the LLM to look at the article form an expert perspective (improving upon ID: 3)	Act as an software analyst and classify the attack from the following choices ...	-21 (57)
7	Citing evidence- asking for evidence from the text (improving upon ID: 3)	Explain your answer using the given definitions and return the option. Give evidence from the article to back up your answer. Use JSON format with the keys: 'explanation', 'option'	-14 (64)

Table 7: Total accuracy over all the articles for each LLM.

Dimension	GPT	BARD
Type of compromise	59%	28%
Intent	88%	88%
Nature	74%	69%
Impacts	52%	45%

these articles (which focus on IT software). That option is data and financial theft, which it chose 49 times out of 65.

We observe that for the articles where the "Type of compromise" (ground truth provided by CNCF), we sometimes agreed with GPT over the CNCF. Figure 5 represents the distribution of GPT's choice and when they were incorrect according to the CNFC ground truth. We examined the 14 articles where both the type of compromise and impacts were incorrectly identified. For these instances, two raters with an inter-rater agreement, κ of 0.82 found that most of the time, if they disagreed with CNCF, they concurred with GPT and vice versa. In the 8 instances where raters disagreed with CNCF, they agreed with GPT 6 times; the same ratio was observed when

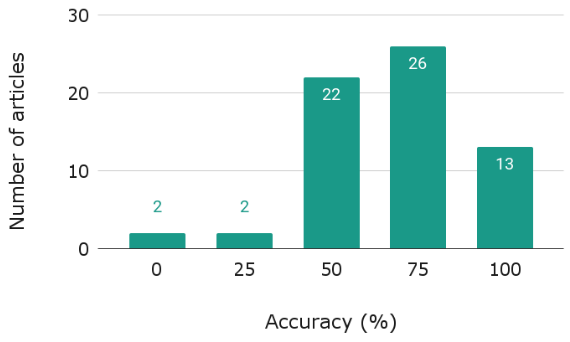

Figure 4: Distribution of the accuracy by articles for GPT. GPT answered 4 questions — so 5 possible outcomes per case.

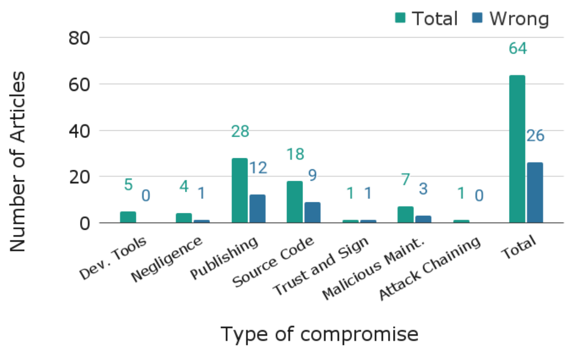

Figure 5: Categorization of articles for the dimension- "Type of Compromise" by GPT. No particular trend is observed.

they disagreed with GPT and agreed with CNCF. For 2/14 articles they disagreed with both GPT and CNCF.

5.2 RQ2: Do LLMs suggest viable mitigation strategies for preventing future failures?

To address our second research question, we asked raters to evaluate GPT's proposed solutions/learnings using a 5-point Likert scale. The average ratings are depicted in Table 8. The mean score across all three questions is 3.83. The raters generally held a positive or neutral view of GPT's "Lessons learned": 42% of the ratings were above 4 (agree), and only 5% of the ratings fell below 2 (disagree).

For further analysis, we randomly selected two articles where the average score of both the raters > 4, and two where < 2. See §10 for the full "Lessons Learned" for these cases.

Factors for strong ratings (average score ≥ 4). We believe the LLM demonstrated good performance in these cases due to the depth of the articles. Article 7 [10] describes the PHP Supply Chain Attack on Pear, and includes technical details of the failures, the exploitation method, and the patch. GPT utilizes the information provided in the blog, combined with its own knowledge, to suggest suitable solutions, *e.g., "encouraging companies and developers to transition from PEAR to Composer"*. Article 35 [47] describes a compromised npm package. It contains technical details of the failure and information on prevention. GPT offers specific solutions, such as *"encouraging the use of Intrinsic or similar Node.js packages to whitelist and control access to sensitive resources and APIs"*.

Factors for weak ratings (average score ≤ 2). We believe the LLM demonstrated poor performance in these cases because the articles had few details. Articles 65 [17] and 67 [77] are brief and lack substantial technical details of the failures. Article 67 discusses remote exploitation of a Gentoo server and mentions ongoing forensics. It primarily serves as a notice to users. Article 65 discusses the backdooring of WordPress but provides little information that could inform solutions/learnings. The advice given by GPT is hence generic, such as *"investigate the incident and address the vulnerability"* and *"conduct code audits"*.

Table 8: Average rater's rating (Likert scale (1-5/"strongly disagree" to "strongly agree") over all the articles of GPT's response to the solution/learnings prompt.

Question	Rating
Is the advice helpful in general for software supply chain failures?	3.72
Is the advice related to the specific failure mentioned in the article?	4.15
Can the advice be used to solve/mitigate the failure mentioned in the article?	3.62

6 DISCUSSION

Is using LLMs worth it in this context? We found that both LLMs in our experiment were capable of simpler forms of analysis, such as distinguishing whether a vulnerability was actually exploited. However, for more complex questions that require some amount of context or judgment, neither LLM achieved a high level of agreement with the CNCF analysts or our manual raters. We believe the current generation of off-the-shelf LLMs does not offer a high enough level of agreement with expert judgment to make it a useful assistant in this context.

One potential path to improving performance is fine-tuning the LLM using baseline knowledge such as this catalog, and then applying it on future issues [22]. An alternative is to integrate a domain-specific NLP model fine-tuned on cybersecurity data (*e.g.,* CyBERT [76]), which might help on this specialized task. We emphasize that CyBERT is *not* an LLM with general Question-Answer capability, so integrating it would require non-trivial design and implementation work.

Will LLMs be a viable alternative to manual analysis in the future? In the past few years, OpenAI's GPT models have advanced from simple tasks (GPT-1, GPT-2) to the performance reported here (GPT-3.5). The recent GPT-4 model is more impressive still [26]. We expect the next generation of LLMs will be suitable aids or replacements for this class of manual analysis.

Future Work. The scope of this analysis could be broadened to encompass additional LLMs, such as Claude [5] and Cohere [15], as well as to incorporate cybersecurity-specific NLP tools such as CyBERT [76]. Additional prompt engineering, and tailoring the prompts per LLM, might improve the accuracy of the results. Lastly, the analysis could be extended to include a wider range of articles and failures beyond those found in the CNCF catalog [2, 3].

7 THREATS TO VALIDITY

Internal: Prompt engineering was conducted with only one of the LLMs (ChatGPT) utilizing literature from its parent organization (OpenAI); the same prompts were used with the other LLM (BARD). The performance of BARD as reported in our study might be misrepresented due to this bias in prompt engineering. Additionally, we relied on manual analysis as the ground truth for our evaluation. We used multiple raters reaching agreement to mitigate bias. We measured an average inter-rater agreement of $\kappa = 0.6$, indicating that independent judgments were generally consistent.

Several issues were identified with the catalog and its articles. (1) Three articles were inaccessible due to broken URLs or PDF formats that were incompatible with LLMs, and were excluded from the analysis [11, 27, 45]. (2) Three articles [21, 77, 80] announced a failure, but no analysis — too little information to answer our RQs. (3) Some of the CNCF article labels did not match the CNCF taxonomy. For example, Article 56 [57] was categorized as a "Fake toolchain", and Article 63 [96] was labeled as a "Watering-hole attack". (4) One article [18] was not relevant.

Bard's low performance could be due to methodological bias. We could not conduct methodological prompt engineering on Bard due to a lack of literature on it. Furthermore, as the API for Bard is currently not out, the number of queries allowed were inadequate to conducted prompt engineering. Therefore, we used available guidance for GPT. Bard's limit of 2000 tokens per prompt was below some prompt lengths, potentially reducing accuracy.

External: Constructed prompts could be over-fitted to analysis in the catalog. Replication of the catalog might not represent failure analysis of incidents in practice. Replication of a single catalog might not generalize to all incidents.

8 CONCLUSION

We evaluate the ability of Large Language Models (LLMs) at characterizing software supply chain failures. Our study revealed that LLMs are particularly effective when manual analysts are able to reach a consensus on the characteristics of the failure. In contrast, their performance tends to deteriorate when the agreement among raters is low. The quality of the LLMs' outputs also depends on the level of detail provided in the source articles, with more comprehensive articles leading to higher-quality responses. We conjecture that while LLMs offer a valuable tool for rapidly analyzing large volumes of text, they have not yet reached a stage where they can replace human analysts or manual classification. Rather than viewing LLMs as a replacement for human input, they should be considered as a supplementary tool that can assist human analysts. As the depth of detail in postmortems and articles increases, and as LLMs continue to improve, they may evolve into viable analytical resources

9 ACKNOWLEDGMENTS

This work was supported by Cisco, Google, and NSF award #2229703.

Per the Association for Computing Machinery's policies on authorship: OpenAI's ChatGPT model (v4) was used during manuscript preparation. Prompt: *Can you make the following clearer? "TEXT SNIPPET"*. We reviewed answers to ensure it did not change the ideas.

10 DATA AVAILABILITY

Our data, analysis, and code is available at https://doi.org/10.5281/zenodo.8365116.

REFERENCES

[1] C. J. Alberts, A. J. Dorofee, R. Creel, R. J. Ellison, and C. Woody. A Systemic Approach for Assessing Software Supply-Chain Risk. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–8, Jan 2011.

[2] P. C. Amusuo, A. Sharma, S. R. Rao, A. Vincent, and J. C. Davis. Reflections on software failure analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, pages 1615–1620, New York, NY, USA, Nov. 2022. Association for Computing Machinery.

[3] D. Anandayuvaraj and J. C. Davis. Reflecting on Recurring Failures in IoT Development. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, pages 1–5, New York, NY, USA, Jan. 2023. Association for Computing Machinery.

[4] D. Anandayuvaraj, P. Thulluri, J. Figueroa, H. Shandilya, and J. C. Davis. Incorporating failure knowledge into design decisions for iot systems: A controlled experiment on novices. In *5th International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT 2023)*, 2023.

[5] Anthropic. Introducing claude. https://www.anthropic.com/index/introducing-claude,, 2023. Accessed: 2023-07-06.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[7] V. R. Basili. The experience factory and its relationship to other improvement paradigms. In *European Software Engineering Conference*, pages 68–83. Springer, 1993.

[8] K. Beck. *Extreme Programming Explained: Embrace Change.* addison-wesley professional, 2000.

[9] M. Bishop. Vulnerabilities analysis. In *Proceedings of the Recent Advances in intrusion Detection*, pages 125–136. Citeseer, 1999.

[10] T. Chauchefoin. Php supply chain attack on pear. https://blog.sonarsource.com/php-supply-chain-attack-on-pear/, 2022.

[11] S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R.-P. Weinmann, E. Rescorla, and H. Shacham. A systematic analysis of the juniper dual ec incident. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 468–479, New York, NY, USA, 2016. Association for Computing Machinery.

[12] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu. Towards intelligent incident management: Why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1487–1497. ACM, 2020.

[13] CNCF Security Technical Advisory Group. Catalog of supply chain compromises. https://github.com/cncf/tag-security/tree/main/supply-chain-security/compromises, 2023. GitHub repository.

[14] J. Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[15] I. Cohere Technologies. Cohere, 2023. Accessed: 2023-07-06.

[16] B. Collier, T. DeMarco, and P. Fearey. A defined process for project post mortem review. *IEEE software*, 13(4):65–72, 1996.

[17] Corbet. The backdooring of wordpress. https://lwn.net/Articles/224997/, 2007.

[18] Corbet. kernel.org status: hints on how to check your machine for intrusion. https://lwn.net/Articles/461237/, 2011.

[19] S. H. Costello. Software engineering under deadline pressure. *ACM SIGSOFT Software Engineering Notes*, 9(5):15–19, 1984.

[20] Cybersecurity and Infrastructure Security Agency. Defending against software supply chain attacks. Technical report, Cybersecurity and Infrastructure Security Agency, April 2021.

[21] G. A. Database. Malicious package in load-from-cwd-or-npm. https://github.com/advisories/GHSA-jxf5-7x3j-8j9m, 2020.

[22] J. C. Davis, P. Jajal, W. Jiang, T. R. Schorlemmer, N. Synovic, and G. K. Thiruvathukal. Reusing deep learning models: Challenges and directions in software engineering. In *Proceedings of the IEEE John Vincent Atanasoff Symposium on Modern Computing (JVA'23)*, 2023.

[23] T. Dingsøyr, F. O. Bjørnson, and F. Shull. What Do We Know about Knowledge Management? Practical Implications for Software Engineering. *IEEE Software*, 26(3):100–103, 2009.

[24] R. J. Ellison, J. B. Goodenough, C. B. Weinstock, and C. Woody. Evaluating and Mitigating Software Supply Chain Security Risks. Technical report, Carnegie-Mellon Univ. Pittsburgh PA Software Engineering Inst., May 2010. Section: Technical Reports.

[25] Enduring Security Framework. Securing the Software Supply Chain: Recommended Practices Guide for Developers. Technical report, Cybersecurity and Infrastructure Security Agency, Aug. 2022.
[26] N. Epson. From GPT-1 to GPT-4: The Evolution of Large Language Models, May 2023. Section: Artificial Intelligence Development.
[27] ErCiccione. Warning: The binaries of the cli wallet were compromised for a short time. https://web.getmonero.org/2019/11/19/warning-compromised-binaries.html, 2019.
[28] M. D. Ernst. Natural Language is a Programming Language: Applying Natural Language Processing to Software Development. In B. S. Lerner, R. Bodík, and S. Krishnamurthi, editors, *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, volume 71 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.
[29] European Union Agency for Cybersecurity. ENISA Threat Landscape 2021. Report/Study, European Union Agency for Cybersecurity, Oct. 2021.
[30] European Union Agency for Cybersecurity. ENISA threat landscape for supply chain attacks. Technical report, Publications Office, LU, July 2021.
[31] European Union Agency for Cybersecurity. ENISA threat landscape for supply chain attacks. Technical report, European Union Agency for Cybersecurity, July 2021.
[32] European Union Agency for Cybersecurity. ENISA threat landscape 2022. Technical report, Publications Office, LU, 2022.
[33] European Union Agency for Cybersecurity. Good Practices for Supply Chain Cybersecurity. Technical report, European Union Agency for Cybersecurity., 2023.
[34] M. E. Fagan. Inspecting software design and code. *Datamation*, 23(10):133, 1977.
[35] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 38(2.3):258–287, 1999.
[36] P. Fonseca, K. Zhang, X. Wang, and A. Krishnamurthy. An Empirical Study on the Correctness of Formally Verified Distributed Systems. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 328–343. ACM, 2017.
[37] V. Garousi, S. Bauer, and M. Felderer. NLP-assisted software testing: A systematic mapping of the literature. *Information and Software Technology*, 126:106321, 2020.
[38] J. I. Gavara. Claude vs chatgpt. *Medium*, 2023.
[39] D. Geer, B. Tozer, and J. S. Meyers. For good measure: Counting broken links: A quant's view of software supply chain security. *USENIX; Login*, 45(4), 2020.
[40] Z. Ghahramani. Lamda: our breakthrough conversation technology. https://blog.google/technology/ai/lamda/, 2023. Accessed: 2023-06-29.
[41] T. Gilb and D. Graham. *Software Inspections*. Addison-Wesley Reading, Masachusetts, 1993.
[42] R. Gill. What is open-source intelligence? = https://www.sans.org/blog/what-is-open-source-intelligence/, 2023. Accessed: 2023-06-21.
[43] B. Gokkaya, L. Aniello, and B. Halak. Software supply chain: review of attacks, risk assessment strategies and security controls.
[44] Google Cloud. Software supply chain security | google cloud. = https://cloud.google.com/software-supply-chain-security/docs/overview, 2023. Accessed: 2023-06-19.
[45] M. Graham. Context threat intelligence — the monju incident. https://www.contextis.com/en/blog/context-threat-intelligence-the-monju-incident, 2014.
[46] J. Huddleston, P. Ji, S. Bhunia, and J. Cogan. How vmware exploits contributed to solarwinds supply-chain attack. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 760–765. IEEE, 2021.
[47] T. Hunter II. Compromised npm package: event-stream. https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502, 2018.
[48] IEEE Standards Association. IEEE Standard for Software Quality Assurance Processes. Technical report, Institute of Electrical and Electronics Engineers, 2014.
[49] International Organization for Standardization. ISO 9001: Quality management systems-requirements. Standard, International Organization for Standardization, 2015.
[50] International Organization for Standardization. ISO/IEC/IEEE 90003:2018 Software engineering — Guidelines for the application of ISO 9001:2015 to computer software. Standard, International Organization for Standardization, 2018.
[51] C. Jones and O. Bonsignour. *The economics of software quality*. Addison-Wesley Professional, 2011.
[52] Kaspersky. Shadowhammer: Malicious updates for asus laptops. https://www.kaspersky.com/blog/shadow-hammer-teaser/26149/, 2019. Accessed: 2023-06-18.
[53] J. S. Ken Schwaber. *The Scrum Guide*. Scrum.org, 2020.
[54] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes. Time Pressure in Software Engineering: A Systematic Review. *Information and Software Technology*, 121:106257, 2020.
[55] P. Ladisa, H. Plate, M. Martinez, and O. Barais. Sok: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1509–1526, Los Alamitos, CA, USA, may 2023. IEEE Computer Society.
[56] N. G. Leveson. *Safeware: System Safety and Computers*. ACM, 1995.
[57] J. Leyden. Apple cleans up ios app store after first big malware attack. https://www.theregister.com/2015/09/21/xcodeghost_apple_ios_store_malware_zapped/, 2015.
[58] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, 2021.
[59] A. Mandour. Gpt-3.5 model architecture. https://iq.opengenus.org/gpt-3-5-model/, 2023. Accessed: 2023-06-27.
[60] J. Manyika. An overview of bard: an early experiment with generative ai. Technical report, Google AI, 2023.
[61] M. Melo and G. Aquino. The pathology of failures in iot systems. In *Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part IX 21*, pages 437–452. Springer, 2021.
[62] National Research Council et al. *Software for dependable systems: Sufficient evidence?* National Academies Press, 2007.
[63] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 736–747, Raleigh North Carolina USA, Oct. 2012. ACM.
[64] C. Nissen, J. E. Gronager, R. S. Metzger, and H. Rishikof. Deliver uncompromised: A strategy for supply chain security and resilience in response to the changing character of war. Technical report, MITRE CORP MCLEAN VA, 2018.
[65] M. Ohm, H. Plate, A. Sykosch, and M. Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In C. Maurice, L. Bilge, G. Stringhini, and N. Neves, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43, Cham, 2020. Springer International Publishing.
[66] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis. Sok: Analysis of software supply chain security by establishing secure design properties. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, SCORED'22, page 15–24, New York, NY, USA, 2022. Association for Computing Machinery.
[67] OpenAI. Gpt best practices. https://platform.openai.com/docs/guides/gpt-best-practices, 2023.
[68] OpenAI. Openai platform. https://platform.openai.com/docs/api-reference/chat, 2023. Accessed: 2023-07-05.
[69] OpenAI. Openai platform - gpt-3.5 models. https://platform.openai.com/docs/models/gpt-3-5, 2023. Accessed: 2023-06-27.
[70] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290. IEEE, 2015.
[71] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci. Vulnerable open source dependencies: counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, Oulu Finland, Oct. 2018. ACM.
[72] K. Pedersen. Barriers for post mortem evaluations in systems development. In *UKAIS Conference, Glasgow, UK. ; Conference date: 19-05-2010*, 2004.
[73] H. Petroski et al. *Design paradigms: Case histories of error and judgment in engineering*. Cambridge University Press, 1994.
[74] S. Pichai. An important next step on our ai journey. https://blog.google/technology/ai/bard-google-ai-search-updates/, 2023. Accessed: 2023-07-03.
[75] S. E. Ponta, H. Plate, and A. Sabetta. Detection, assessment and mitigation of vulnerabilities in open source dependencies. *Empirical Software Engineering*, 25(5):3175–3215, Sept. 2020.
[76] P. Ranade, A. Piplai, A. Joshi, and T. Finin. Cybert: Contextualized embeddings for the cybersecurity domain. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3334–3342. IEEE, 2021.
[77] D. Robbins. Gentoo linux security announcement 200312-01. https://archives.gentoo.org/gentoo-announce/message/7b0581416ddd91522c14513cb789f17a, 2003.
[78] I. Sarwar, A. Samad, and S. Mumtaz. Object oriented software modeling using nlp based knowledge extraction. *European Journal of Scientific Research*, 35:22–33, 01 2009.
[79] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. F. C. Uribe, and L. Fedus. Introducing chatgpt. https://openai.com/blog/chatgpt, 2023. Accessed: 2023-07-03.
[80] E. Schwartz. [aur-general] acroread package compromised. https://lists.archlinux.org/pipermail/aur-general/2018-July/034152.html, 2018.
[81] Security Technical Advisory Group. Software Supply Chain Best Practices. Technical report, Cloud Native Computing Foundation, May 2021.
[82] I. Sommerville. *Software Engineering*, volume 137035152. Pearson Education, 2015.
[83] Sonatype. State of the software supply chain. https://www.sonatype.com/resources/state-of-the-software-supply-chain-2021, 2021.
[84] Sonatype. State of the Software Supply Chain. Technical Report 8th Annual, Sonatype, 2022.

[85] Synopsys. 2023 OSSRA Report. https://www.synopsys.com/software-integrity/engage/ossra/rep-ossra-2023-pdf.

[86] The Linux Foundation. SLSA: Supply-chain levels for software artifacts. https://slsa.dev, 2022. Accessed: 2022-04-30.

[87] The Recorded Future Team. What is open source intelligence and how is it used? = https://www.recordedfuture.com/open-source-intelligence-definition, 2022. Accessed: 2023-06-21.

[88] N. Vasilakis, A. Benetopoulos, S. Handa, A. Schoen, J. Shen, and M. C. Rinard. Supply-Chain Vulnerability Elimination via Active Learning and Regeneration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1755–1770, Virtual Event Republic of Korea, Nov. 2021. ACM.

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

[90] K. Vijayakumar and C. Arun. Automated risk identification using NLP in cloud based development environments. *Journal of Ambient Intelligence and Humanized Computing*, 2017.

[91] K. Vivek. Is software reuse leading to dependency hell? = https://www.linkedin.com/pulse/software-reuse-leading-dependency-hell-vivek-kant, Sept. 2022.

[92] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

[93] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839*, 2023.

[94] M. Yao. Top 6 nlp language models transforming ai in 2023. *TOPBOTS*, 2023.

[95] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams. What are Weak Links in the npm Supply Chain? In *International Conference on Software Engineering (ICSE)*, 2022.

[96] K. Zetter. 'google' hackers had ability to alter source code. https://www.wired.com/2010/03/source-code-hacks/, 2010.

[97] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro. Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. *ACM Computing Surveys*, 54(3):1–41, 2022.

[98] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen. A survey of large language models, 2023.

[99] M. Zimmermann, C.-A. Staicu, and M. Pradel. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *USENIX Security Symposium*, 2019.