mathematics

MDPI

Article

Construction of Software Supply Chain Threat Portrait Based on Chain Perspective

Maoyang Wang [1], Peng Wu [2,*] and Qin Luo [1]

[1] School of Computer Science, Southwest Petroleum University, Chengdu 610500, China; 202221000497@stu.swpu.edu.cn (M.W.); luoqin@swpu.edu.cn (Q.L.)
[2] School of Information and Engineering, Sichuan Tourism University, Chengdu 610100, China
* Correspondence: 0001524@sctu.edu.cn

Abstract: With the rapid growth of the software industry, the software supply chain (SSC) has become the most intricate system in the complete software life cycle, and the security threat situation is becoming increasingly severe. For the description of the SSC, the relevant research mainly focuses on the perspective of developers, lacking a comprehensive understanding of the SSC. This paper proposes a chain portrait framework of the SSC based on a resource perspective, which comprehensively depicts the threat model and threat surface indicator system of the SSC. The portrait model includes an SSC threat model and an SSC threat indicator matrix. The threat model has 3 levels and 32 dimensions and is based on a generative artificial intelligence model. The threat indicator matrix is constructed using the Attack Net model comprising 14-dimensional attack strategies and 113-dimensional attack techniques. The proposed portrait model's effectiveness is verified through existing SSC security events, domain experts, and event visualization based on security analysis models.

Keywords: software supply chain; software supply chain threat model; attack technique matrix; software supply chain portrait

MSC: 68M25

Citation: Wang, M.; Wu, P.; Luo, Q. Construction of Software Supply Chain Threat Portrait Based on Chain Perspective. *Mathematics* **2023**, *11*, 4856. https://doi.org/10.3390/math11234856

Academic Editor: Shaomin Wu

Received: 13 October 2023
Revised: 28 November 2023
Accepted: 29 November 2023
Published: 2 December 2023

1. Introduction

With the development of the software industry, the Software Supply Chain (SSC) has become increasingly complex. The SSC is a series of processes that begin with the selection and acquisition of software components from third parties through development, distribution, and utilization, it is susceptible to substantial risk due to its comprehensive coverage of the software lifecycle. Software Supply Chain Attacks (SSCAs) refer to an attack against anything on the SSC that the end target depends on. Due to low cost and high payoff potential, SSCAs have become popular with attackers and the preferred tool for nation-state APT attacks. Notably, the SolarWinds event [1] impacted more than 18,000 customers, including government agencies and critical vendors. The CCleaner incident [2] had a significant impact on millions of users, as malicious codes were involved. Furthermore, SSCAs are currently exhibiting a noteworthy trend of rapid growth. For instance, ReversingLabs [3] reported a staggering 289% increase in attacks on npm and PyPI libraries over the past four years. In 2022, nearly 7000 malicious packages were discovered on npm, representing a surge of over 9000% from 2020 and a 40% increase from 2021. Similarly, the number of malicious PyPI packages grew by over 18,000% compared to 2020. Additionally, according to Gartner's [4] predictions, by 2025, 45% of organizations will have experienced SSCAs.

Supply chain threats have become a significant area of research, with scholars proposing multiple strategies from management, detection, and countermeasures viewpoints. However, one of the most efficient methods to tackle this challenge is by accurately and comprehensively defining the SSC threats surface. Zhenfei et al. [5] abstracted the SSC as a consolidated system comprising three key phases: development, delivery, and utilization.

Du et al. [6] defined the SSC as a chain that spans the entire software lifecycle, including software development, inheritance deployment, upgrade, repair, and final distribution. Most of the aforementioned research is abstracted from the perspective of the software lifecycle or the developer's viewpoint, focusing more on the traditional models of software development, maintenance, and delivery. However, such abstractions do not provide a complete description of the supply chain, and they overlook the fact that the SSC, with the addition of 'open source' components and diversified software delivery methods, is not merely a single model from the traditional development perspective. With the widespread adoption of technologies such as cloud services, open-source middleware, and containers, the SSC has become diverse and complex, with new third-party participants at each stage. Therefore, there is a need for a deeper understanding of the current state of the SSC, identification of entry points, a comprehensive grasp of attackers' objectives and methods, and based on this, the construction of an SSC threat portrait that encompasses more critical phases and provides a detailed description of attacker tactics.

This paper proposes a method to build an SSC threat portrait model based on the chain perspective, mainly including the software supply chain model(SSCM) and software supply chain indicator matrix(SSCIM). The analysis is primarily based on SSC events using the ChatGPT [7], complemented by human experiences to deeply explore the common characteristics of the SSC itself and the attacks. Based on this analysis, a three-layered model for SSC is developed, taking a chain-based perspective. Following this, an SSCA indicator matrix is created, using guidance from the Attack Net [8] and MITER ATT&CK [9] attack models. Finally, the effectiveness of the threat portrait model was verified through numerous events and expert assessments, and some events were excerpted for visual display.

The novelty of this paper is threefold:

- An analytic framework for a comprehensive analysis of the SSC portrait is proposed. The framework comprises three core stages: event collection, event analysis, and model construction, alongside seven specific processes. It introduces an innovative, whole-process analytical methodology for constructing the SSC threat portrait.
- We propose an innovative method of event analysis that combines a generative artificial intelligence model, using artificial intelligence and human experience, to deeply mine key nodes and chains in SSC events.
- The overall model and micro-level attack indicators offer a comprehensive and accurate depiction of the SSC threat surface. The model comprehensively characterizes the supply chain from a macro and chain-oriented perspective with 3 levels and 31 dimensions. This broadens the research perspective and accurately captures the interconnectedness and interdependency within the entire supply chain. The attack indicators, guided by attack techniques, present a detailed description with 14 tactical and 113 technical dimensions, accurately describing attack behavior in the supply chain.

The above-mentioned research accurately analyzes the supply chain and its potential threats, providing research value and innovation.

2. Related Work

The SSCAs have been occurring regularly, with instances such as SolarWinds [1] and Log4j [10] having major consequences. This has made SSCAs a focus of research, with relevant academics diving into themes such as the supply chain threat situation, attack detection, and defense.

2.1. Software Supply Chain Threat Surface

Researchers have presented several threats from diverse angles in response to potential threat surfaces inside SSC. Zhou et al. [5] highlighted two important components of SSC threats: employing deception and tampering to contaminate software in the supply chain and exploiting contaminated software to obtain access to target environments, potentially leading to data breaches. Torres-Arias et al. [11] and Peterson et al. [12] investigated new

threats in the context of open-source SSC. They highlighted threats such as unintended or malicious vulnerability introductions in open-source components, the inclusion of harmful logic into open-source components, and the chance of supplier trust credentials being stolen. Ji et al. [13] offered a thorough and comprehensive assessment of SSC threat models from an open-source viewpoint, as well as in-depth talks regarding attack pathways and their consequences. Benthall, S. et al. [14] connected public vulnerability databases with open-source project version control data to evaluate SSC threat risks using publicly available data. Pfretzschner B et al. [15] examined weaknesses in Node.js and four related threat surfaces, suggesting a method to detect dependency-based Node.js attacks. Gokkaya, B. et al. [16] studied typical supply chains to identify the latest trends in SSCAs, emphasizing vulnerabilities connected with third-party components and open-source software supply chains while proposing novel mitigation techniques. Liu, C. et al. [17] presented a knowledge graph-based dependency resolution approach that parses dependency connections into dependency trees and performs large-scale studies into vulnerabilities and security danger surfaces inside these dependency trees. Zimmermann, M. et al. [18] investigated security threats surfaces within the npm ecosystem by systematically analyzing dependency relationships, maintainers, and publicly reported security concerns across software packages.

To address security vulnerabilities in open-source software packages, scholars have conducted an extensive analysis of numerous packages to identify potential threats. Marc Ohm et al. [19] offered a dataset of open-source software packages with a focus on the SSC. They manually analyzed and collected 174 malware packages, furthering the work of the open-source and research communities in detecting and preventing attacks in the SSC. In a study of 10,000 JavaScript npm packages, Nusrat Zahan et al. [20] analyzed metadata to identify six signals of security vulnerabilities in the SSC that can predict high-risk packages for supply chain attacks. Dey, T. et al. [21] examined how software dependency networks impact JavaScript package downloads and concluded that changes in downloads are closely related to the characteristics and number of dependent packages. In response to the possible threats caused by open-source collaboration, scholars have studied open-source communities and open-source libraries. Gonzalez et al. [22] conducted malicious submission attacks on GitHub and pointed out that there are malicious submissions or malicious deletion operations in open-source projects. The purpose of these operations is to inject malicious code into the project or cause damage. Duan Ruian et al. [23] proposed a comparative framework for package manager security assessment to qualitatively evaluate the functionality and security features of language package managers. Wei Tang et al. [24] proposed LibDX, a fully automatic system that can overcome compilation diversity among binaries for detecting reused libraries in binaries and is platform-independent.

2.2. Software Supply Chain Attack and Threat Detection

Regarding SSCA techniques and threat intelligence research, researchers have summarized a variety of effective methods. Ladisa, P. et al. [25] proposed a classification method for open-source supply chain attacks, covering 107 attack vectors in the form of an attack tree. Bos et al. [26] discussed and summarized SSCAs in package language managers. Reed, M. et al. [27] and MITER [28] studied the overall view of SSCAs and summarized the framework and attack modes of attacks. Buchicchio et al. [29] studied SSCAs in Java language, proposed a new type of supply chain attack and new vulnerabilities, and proposed preventive measures. Neil, L. et al. [30] mine threat intelligence about open-source projects and libraries from reported bugs and issues on public code repositories, track the library and project dependencies of installed software on client computers, and display them through a security knowledge graph. Neupane, S. et al. [31] studied a dataset of more than 1200 package obfuscation attacks, strictly defined the obfuscation mechanism of 13 types of SSC package obfuscation attacks, proposed an attack detector, verified and evaluated it on the npm package set. Zahan, N. [32] studied software security frameworks and attack vectors and identified and evaluated security indicators that can be used to detect risky components in dependencies.

For SSC threat detection, researchers have proposed different detection methods. Marc Ohm et al. [33] proposed Buildwatch, a software and third-party dependency detection framework based on dynamic analysis of attack cases to detect possible threats caused by third-party dependencies in the SSC. Wu Zhenhua et al. [34] summarized the research status of program reverse analysis technology in SSC pollution detection problems, proposed a pollution classification model, and proposed a reliable incremental software automation installation method. They also proposed a method to detect software bundling based on software installation information and machine learning techniques. Duc Ly Vu et al. [35] analyzed 2666 PyPI packages and proposed using source code repositories such as Github to detect malicious code injection from third-party packages. Wang [36] summarized the conditions for effectively preventing attacks, confirmed the feasibility of detecting certain SSCAs, and proposed an attack detection method based on information flow.

3. The Framework of Software Supply Chain Threat Portrait Model

3.1. Software Supply Chain Threat Portrait Model Design Framework

This paper presents a framework model for conducting a threat analysis of SSC, which comprises three layers: event collection layer, event analysis layer, and model construction layer. The architecture of the model is illustrated in Figure 1. The event collection layer is responsible for collecting detailed descriptions and technical analysis texts of SSCA incidents, while the event analysis layer assesses the overall SSC and attack technical details. On the basis of the first two layers and relevant theory, a model and an attack indicator matrix are established, and then the SSC threat portrait is constructed.

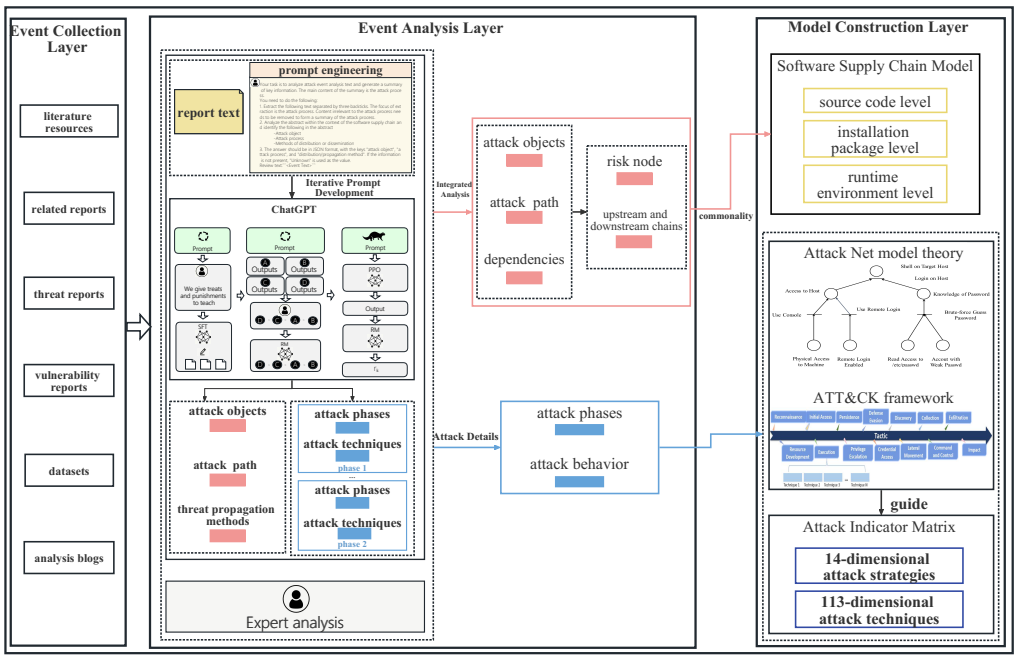

Figure 1. SSC threat portrait model design framework.

3.2. Event Collection Layer

Event analysis, verification, and threat analysis models are built on the basis of SSCA events. This paper uses vocabulary related to SSC and SSCA events as the subject words for event collection, crawls related websites using the scrapy and selenium frameworks based on Python language, and combines a linkage-based approach with content evaluation as the search strategy. The linkage-based approach starts from links that are highly related to the subject words, expanding the collection scope through closely related links, such as friendly links and reference links on the pages. In this process, the collection scope is mainly limited to links that are strongly related to cybersecurity and SSC. The strategy

based on content evaluation starts from the relevance between the content of the link page and the subject words, considers the similarity of the title, text, and topic, and selects pages strongly related to the keywords as collection results. In this process, the content collected is mainly page content closely related to SSCAs.

3.3. Event Analysis Layer

This layer identifies potential risk objects and attack routes, as well as downstream and upstream dependencies in the SSC. It considers factors like attack targets, attack processes, and methods of threat propagation. Its focus is on initiating from the attack phases and techniques with an attack-oriented perspective to elucidate attack indicators in SSCAs. This prepares for building subsequent threat portrait models.

Large Language Models (LLMS) such as ChatGPT are rapidly evolving and have been attempted to be applied to cybersecurity threat analysis. Purba et al. [37] proposed a text mining methodology and improved it based on ChatGPT to analyze actionable cyber threat intelligence from Twitter streams. Wang et al. [38] used ChatGPT to extract objects and attributes from textual data in the threat intelligence domain and perform data filtering and mapping to reconstruct threat intelligence using semantic characteristics of events, which improves the efficiency aggregation and automated analysis level of threat intelligence. Perrina et al. [39] proposed a system AGIR incorporating ChatGPT to automate the generation of comprehensive intelligence reports using two different natural language generation methods to extract key information from threat reports. Fayyazi et al. [40] used LLMs such as ChatGPT to analyze ambiguous cyberattack description texts and mapped them to MITRE ATT&CK tactics. In addition, LLMs have been applied to intrusion detection [41], program analysis [42], code analysis [43], and assisting threat intelligence analysis with natural language processing (NLP) algorithms [44–46]. Recent studies indicate that LLMs perform well in diverse downstream tasks without parameter tuning, even requiring only a few examples as instructions. He et al. [47] proposed an in-context learning framework and introduced formatting demonstrations to extract information from text, with the framework performing much better than previous pre-trained methods on all datasets. Wei et al. [48] conducted zero-shot information extraction via LLMs, such as ChatGPT, and the results show that ChatGPT achieves impressive performance and even surpasses some full-shot models on several datasets.

This paper utilizes ChatGPT [7] to extract the key information of the attack from the text of the event description and uses zero-shot learning based on in-context learning (ICL) to construct the prompt templates. Next, the expert experience is utilized to confirm and filter the answers of ChatGPT and summarize the key information, nodes, and relationship chains regarding the construction of the SSCM and SSCIM. To ensure that ChatGPT generates answers that meet specific task requirements, it is essential to follow these strategies: use delimiters, structure the output, model self-checks to verify that conditions are met, provide a few example prompts, specify the task steps, and ask the model to prioritize autonomous problem-solving strategies before reaching any conclusions. This paper sets up two different prompt texts through multiple prompt iterations. Additionally, the 'Review text' represents the analysis text of a certain attack event, as shown in Figures 2 and 3.

Through manual analysis and the use of the ChatGPT language model, the key information, nodes, and relationship chains regarding the construction of the SSCM and SSCIM were summarized. Tables 1 and 2 below show excerpts of some typical incident analysis cases.

Table 1. Overall analysis of events for building SSCM.

Event	Risk		
	Attack Objects	**Attack Path**	**Upstream and Downstream Chains**
XcodeGhost [49]	Software code Software installation package	Attackers implant malicious code into the development tool XcodeGhost and spread it maliciously through cloud services and resource sharing.	Software code → Development tool Software installation package → resource sharing, cloud services
Event-Streams [50]	Software code	Malicious maintainers insert malicious code into third-party component code	1. For EventStreams: Software code → Developers 2. Affected downstream code: software code → third-party components → third-party open source library

Overall Analysis of SSC - ChatGPT Prompt Text

8

```
Your task is to analyze attack event analysis text and generate a summary of key
information. The main content of the summary is the attack process.
You need to do the following:
1. Extract the following text separated by three backticks. The focus of extraction is
the attack process. Content irrelevant to the attack process needs to be removed to form
a summary of the attack process.
2. Analyze the abstract within the context of the software supply chain and identify the
following in the abstract:
                -Attack object
                -Attack process
                -Methods of distribution or dissemination
3. The answer should be in JSON format, with the keys "attack object", "attack process",
and "distribution/propagation method". If the information is not present, "Unknown" is
used as the value.

Review text:```<Event Text>```
```

Figure 2. Overall analysis of events for the SSCM construction.

3.4. Model Construction Layer

The SSC threat portrait model consists of two parts: a Software Supply Chain Model (SSCM) constructed from an overall perspective and a Software Supply Chain Indicator Matrix (SSCIM) constructed from an attack perspective.

The Software Supply Chain Model (SSCM) is a part of the threat portrait proposed in this paper. The SSCM depicts the SSC from a macro perspective, encompassing the dependency chains of four different aspects of the software: the Source Code, the Components, the Runtime Environment, and the Installation Package. At the same time, the nodes in the model can not only serve as upstream traceability nodes for software in the supply chain but also serve as possible attack penetration points. Based on the characteristics of the upstream and downstream dependency chains of the complex SSC, a three-layer model is constructed from the perspective of the chain. From the core to the outermost layer, the model can be viewed as the traceability process of the supply chain from downstream to upstream.

Based on the above characteristics, this paper defines SSCM as a quintuple consisting of Source Code, Installation Package, Runtime Environment, Components, and Attack Techniques:

$$M = T(A, I, C, E, S) = A(I) \times A(C) \times A(E) \times A(S) \quad (1)$$

Table 2. Attack detail analysis for constructing SSCIM.

Events	Attack Phases	Attack Behavior
XcodeGhost	Implant malicious modules	Develop a malicious version of the development tool Xcode
	Malicious propagation	Phishing techniques trick developers into downloading unofficial versions
	Execute malicious commands	Use powerful interfaces and runtime backdoors in the system to issue malicious commands
	Evade defense measures	Install certificates on target systems through social engineering and forced authentication
	Control target systems	Use pseudo-protocol combined with malicious remote control modules to control the target system
	Capture sensitive information	Collect sensitive information such as network information and credentials from controlled systems
Event-Streams	Implant malicious modules	open-source repository maintainers maliciously submit components containing malicious code
	Trigger malicious code execution	Exploit runtime backdoors and trick users into triggering the malicious code execution
	Establish long-term control	Exploiting backdoors in code
	Obtain Higher Privileges	Plant malicious dependencies into the repository of high-privilege users
	Obtain Legal Access	Exploit the compromised credentials

Detailed Analysis of SSCAs - ChatGPT Prompt Text

Your task is to analyze the text of the attack incident report and generate a summary of key information, focusing on attack procedures and techniques.
You need to do the following:
1. Extract the following text separated by three backticks. The extraction focuses on the attack process and the specific attack techniques used by the attacker. Remove any information that does not relate directly to the attack process and techniques, and form a summary of the attack process and attack technology.
2. Using the software supply chain as the background, analyze the summary content and identify the following elements in the summary:
- Attack process
3. Analyze the obtained attack process. If it contains a series of sub-attack processes, please analyze the sub-attack processes and the attack techniques contained in them one by one, and rewrite these processes into the following format:
1 - Attack Phase: ...
Attack techniques: ...
2 - Attack Phase: ...
Attack techniques: ...
...
N - Attack phase: ...
Attack techniques: ...
Do not directly use the attack process explicitly provided in the article until the problem is solved. Make sure you speak clearly and that you can solve the problem yourself.
Find answers in concise sentences.
4. Your answer is in JSON format, with the serial number, "Attack phase" and "Attack techniques" as key values. If the information is not present, "Unknown" is used as the value.
Review text: ```<Event Text>```

Figure 3. Analysis of attack details for the SSCIM construction.

The mathematical notation used in this chapter is shown in Table 3.

Table 3. Definition of mathematical notation in SSCM model.

Mathematical Notation	Definition
C	Components level
S	Source Code level
E	Runtime Environment level
I	Installation Package level
M	Represents the final model obtained
$A(x)$	Represents the attack techniques present on x
$T(y_1, y_2, \ldots, y_n)$	Represents threats present on the y_i

The Software Supply Chain Indicator Matrix (SSCIM) is an additional component of the threat portrait proposed in the paper. It offers a comprehensive depiction of the SSC from an attack perspective. In terms of the construction of SSCIM, based on the network attack model Attack Net [8], the different stages, behaviors, and processes of SSCAs are analyzed. At the same time, referring to the architecture foundation of MITRE ATT&CK [9] and the open framework OSC&R [51], the attacker's attack techniques and behaviors are taken as the focus, and an attack indicator matrix is further refined and constructed in the form of indicator matrix to describe the attack behaviors and techniques comprehensively and systematically. The attack indicator matrix focuses on the characteristics of SSCAs, the tactics correspond to the attack phase, and several technologies under the tactics correspond to the possible attack behaviors. Several technologies under each tactic are connected to form a complete attack technology chain.

Taking XcodeGhost event as an example, first of all, in the stages of "malicious propagation" and "Implant malicious modules", the attackers adopted the attack behaviors of "phishing" and "developing malicious versions of software" to pave the way for the subsequent stages of attacks. Secondly, in the "execute malicious commands" and "evade defense measures" stages, attackers use "utilize system interfaces" and "forced authentication" attack behaviors to evade defense measures. Then in the "control target systems" stage, "use pseudo-protocol technology" is used to control the target system. Ultimately, the "capture sensitive information" phase shows the attacker's ultimate goal. This series of attack stages and behaviors forms a clear attack path and reveals the specific details of the attack process. As shown in Figure 4, "○" represents the attack stage, and "—" represents the attack behavior.

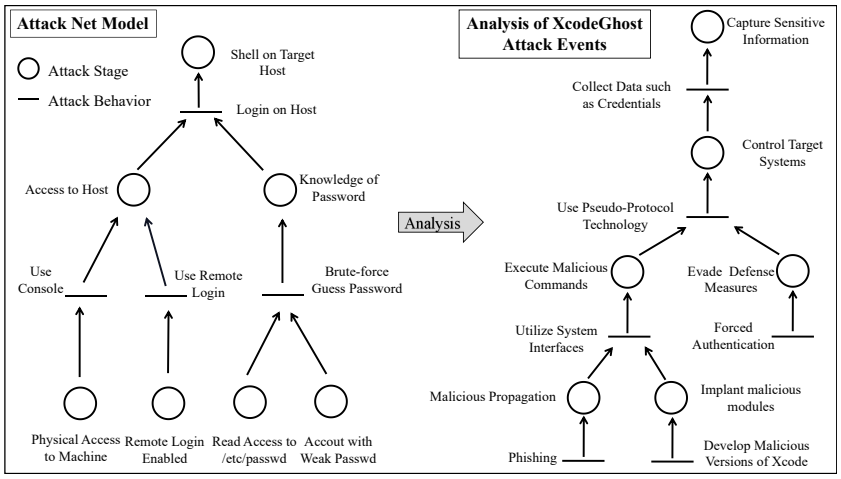

Figure 4. Attack Net model and application in event cases.

4. Software Supply Chain Model

The software supply chain model defined in this paper is an open-source SSC formed by developers and tools, third-party components, software delivery methods, running processes, and other nodes in the dependence relationship of the supply chain, and it is also a network of infiltration points that may have infiltration risks in the supply chain. As shown in Figure 5, this paper summarizes the open-source SSC into three different levels, namely the inner layer, the middle layer, and the outer layer. The further inward the hierarchy is, the stronger the connection between the nodes and the software itself; conversely, the further outward the hierarchy is, the more the nodes within it tend toward the upstream part of the chain. The following will introduce the SSCM in hierarchical order.

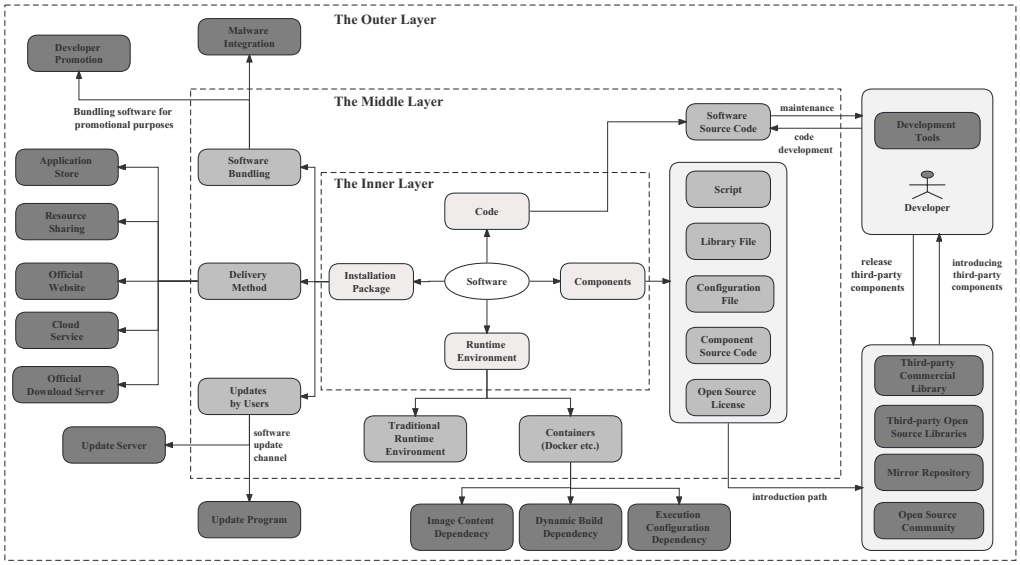

Figure 5. Software supply chain model.

- The Inner Layer

The innermost layer contains the four layers most closely related to the software itself, which are the code level, component level, installation package level, and runtime environment level. These are also different levels at which security threats may exist.

- The Middle Layer

The nodes in the second layer are derived from the nodes in the preceding layer. To begin with, the "Code" node extends the "Software Source Code" node at the code level. Software code is often authored by developers utilizing development tools and subsequently upheld by said developers. Furthermore, the scope of the component level can be broadened to encompass various nodes such as "Component Source Code", "Script", "Library File", "Configuration File", and "Open-Source License". These nodes represent different facets of the component, indicating that components not only consist of the associated source code but also rely on script files for specific functionalities. In addition, library files are a collection of precompiled methods that can be referenced and reused by other programs as another form of component. Simultaneously, there exist interdependencies among components and configuration files. The utilization of configuration files allows for the segregation of parameters from component code, hence enhancing the configurability and flexibility of these components.

Finally, for open-source components, the open-source license clearly stipulates how others can use, modify, and distribute the component, ensuring its reasonable and legal use and dissemination. Thirdly, the installation package acquisition layer can be extended to the "Software Bundling", "Delivery Method", and "Updates by Users" nodes. That is, users can obtain software installation packages from the above three nodes. Fourthly, the "Runtime Environment" can be extended to "Traditional Runtime Environment" and "Container (Docker etc.)" running modes. In other words, when users try to run the software, they can use traditional local operations or cloud-native Docker containers.

- The Outer Layer

The nodes of the third layer are all traced upstream from the nodes of the second layer.

For the outer nodes that extend from the "Code" node: "Developers" collaborate on the development of software source code, utilizing "Development Tools" and maintaining the source code. Developers can also compile and generate third-party components by using development tools and release them to "third-party libraries", or import third-party components into the development environment or software source code from the "Third-

party Open-Source Library", "Third-party Commercial Library", "Mirror repository", or "Open-Source Community" to assist in efficient and convenient development work.

In the process of tracing back from the inner code node to this point, the attacker can use social engineering, steal the developer's private data, and other ways to attack and contaminate the developer's development environment and tools; or developers introduce defective or compromised third-party components into the development environment. This leads to the risk of defects or built-in backdoors in the developed software or components. Since developers are all trustworthy developers or reputable development companies, it is difficult to review malicious code from the source code level, such malicious code is difficult to detect in the short term and can be hidden for a long time.

For the outer layer nodes extended from the "Script" and other five nodes in the middle layer, the "Script" and "Library File" can be introduced into the software by the "Third-party Open-Source Library", "Third-party Commercial Library", "Mirror Repository", or "Open-Source Community" nodes located in the outer layer, the security threats potentially exist in these outer layer nodes. Therefore, elements such as libraries, configuration files, and component source code introduced through these external resources may carry malicious components, introducing potential threats into downstream software products. It is worth noting that most of the current open-source libraries have security vulnerabilities related to open-source licenses, which may lead to legal risks.

For the outer-layer nodes extending from nodes such as "Software Bundling", "Delivery Method", and "Updates by Users": On the one hand, users can obtain packages through different ways, such as downloading packages through official channels such as "Application store", "Official Website", or "Official Download Server" nodes. However, the attacker can attack the official way to contaminate the package or perform obfuscation attacks and impersonation to achieve malicious replacement and dissemination. In addition, "cloud service" and "resource sharing" nodes are also popular ways for users to obtain software packages, such as non-official cloud services like torrent files and cloud drives. These unofficial delivery routes do not follow a set of security regulations and review standards, that is the source of the software package is unknown and has not undergone security checks. An important area of concern is packages modified by unknown third parties, such as "hacked versions", "green versions", and "in-app purchase versions", which are favored by attackers for embedding malicious code and malicious programs.

On the other hand, the middle-layer node of "Software Bundling" refers to software installation packages or compressed files that are bundled with third-party packages or malware. Bundling can be integrating malicious programs into normal software packages, such as hiding malicious programs in green software, that is, "Malware Integration" node. In addition, some developers may bundle their software with other packages due to promotion needs and to reduce marketing costs, which is called "Developer Promotion" node. Such bundled software installs silently in the background, without even an installation prompt. This can easily result in concealed malicious programs, leading attackers to launch attacks on the user's system.

Finally, the normal installation package update channel for users is to obtain it from the official update server or update program, that is, the "update server" and "update program" nodes. For this method of obtaining software packages, attackers can achieve malicious implantation or malicious replacement of software packages by attacking the software update server, update program, and update process.

For the runtime environment: On the one hand, users can run the software in the traditional local runtime environment. Threats may exist in system vulnerabilities, backdoors, library vulnerabilities, or configuration vulnerabilities in the local operating environment. On the other hand, the software can be run directly through cloud-native environments such as docker and other containers. There are three possible dependencies when running a container. First, image content dependency, that is, container images are built on a centralized repository and image-sharing mechanism. Users can upload container images created by themselves and download container images shared by other users; therefore,

there are complex dependencies between images. The dependencies between containers in containerized deployment are complex, and images are often reused, interdependent, and form new chains. Therefore, attackers can infiltrate the entire container cluster by tampering with or inserting malicious code into an image at some node in the complex chain, and the potential risk of third-party components introduced by Docker images during the development process is multiplied through the container image dependency chain, causing a serious threat situation. The second is performing configuration dependencies, where a container image might be based on other images and inherit some configuration information. The downstream images inherit the insecure configuration from the upstream through dependencies. The third is dynamic build dependencies, that is, in order to update the application or deploy a new application in the container, a new image is often generated according to the predefined build instructions through the provided build tool. Such a dynamic build dependency method can improve the efficiency of application deployment. Attackers can exploit this by implanting malicious code into packages to attack containers. It is also possible to fake dependencies and introduce malicious dependencies into the container at build time.

5. Software Supply Chain Attack Indicator Matrix

The indicator matrix proposed in this paper is described in the form of a combination of tactics and techniques. The techniques in the matrix cover the widest range of attack vectors, including third-party libraries, components, and vulnerabilities that often threaten the SSC, helping to understand potential threats in the supply chain and formulate security strategies. There are 14 different tactical dimensions in the attack indicator matrix and 113 technical indicators under the tactical dimension, as shown in Tables 4 and 5. Table 4 shows the first 7 tactical dimensions, and Table 5 shows the last 7 tactical dimensions. Since the same attack techniques may appear in different tactical dimensions, there will be a small number of repeated technical indicators in the matrix.

5.1. Reconnaissance

"Reconnaissance" refers to the first steps a hacker takes within the SSC. The attacker needs to collect valuable and usable information about the target first, so as to carry out more accurate and effective attacks in the subsequent process, so as to effectively promote the attack process and minimize the risk of detection.

Scan Used Open-Source Dependencies: During the reconnaissance phase, attackers can scan and identify open-source dependencies present in the target software, such as third-party open-source libraries, frameworks, and other components introduced in the software, from which attackers can discover potential vulnerabilities and penetration points.

5.2. Resource Development

"Resource Development" is when an attacker establishes some resources for future attacks, such as creating, purchasing, or stealing resources that can be used in targeted attacks. These resources include infrastructure, accounts, and capabilities. Attackers can use these resources to support subsequent attack operations.

Contributing Malicious Code to Open-Source Repository: The upstream open-source repository is used to manage and store open-source project code. In an open-source community, multiple contributors work together to improve and maintain open-source projects through community contributions. Attackers may maliciously submit code that hides malicious components or backdoors during the improvement and maintenance process. Due to people's trust in open-source repositories, subsequent risks are difficult to detect, and the code may be integrated into more downstream projects and software, causing widespread adverse effects.

Table 4. The First 7 Tactical Dimensions of SSCIM.

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion
Scan Used Open-Source Dependencies	Contributing Malicious Code to Open-Source Repository	Code Repository Contamination	Package Manager Threats	Backdoor in Code	Implant Malicious Dependencies in High-Privilege User Repository	Saas Sprawl
Scan Code Repository	Malicious Behavior by Open-Source Maintainers	Development Environment Threats	Executing Malicious Components/Extensions	Implant in Zombie Instance	Malicious Components Introduced in High-Privilege User Environments	Configuration Error
Scan Public Container Images	Malicious Substitution of Legitimate Items	Development Tool Threats	IDE Malicious Build Injection	Deploy Keys	Exploiting Software Vulnerabilities	Development Tool Threats
Investigate Supply Chain Relationships	Develop Malicious Project	Vulnerabilities in Third-Party Components/Extensions	Exploiting Software Vulnerabilities	Browser Extensions	Manipulate Access Token	Exploiting Vulnerabilities
Phishing for Information	Malicious Acquisition	Repojacking	Runtime Backdoor	Manipulate Access Token	Abusing Privilege Escalation Control Mechanisms	Impair Defense Mechanisms or Components
Active Scanning	Hacking the Update Server	Combosquatting	SQL Injection	Implanting Container Images	Exploiting Valid Accounts	Manipulate Access Token
Scan Open Information Sources	Hacking the Official Website	Typosquatting	Command Execution	Abuse Task Scheduling		Exploiting Valid Accounts
Collect Target Host Information	Steal Accounts	Dependency Confusion	Cross-Site Scripting	External Remote Services		Abusing Privilege Escalation Control Mechanisms
Collect Target Identity Information	Create and Cultivate Account	Brandjacking	User Execution	Account Manipulation		Indicator Removal
		Shadow IT	API Exploitation	Establish Accounts		Subvert Trust Controls
		Exposed Public Artifacts or Information	Deploy Container			Alternate Authentication Material Theft
		CI/CD Threats	Abuse of Container Management Commands			Obfuscate Files or Information
		Malicious Module Injection				
		Exploiting Trusted Third-Party Relationships				
		Exploiting Application Vulnerabilities and Errors				
		Phishing				
		Exploiting Valid Accounts				
		Drive-by Compromise				
		External Remote Services				

Table 5. The Last 7 Tactical Dimensions of SSCIM.

Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Steal Third-Party Software Tokens	Discover Dependent Code Repository	Push Implants Across Repositories	Information from Code Repository	Remote Access Software	Exfiltration to Code Repositories	Implant Malicious Module
Brute Force	Discovering System Information	Exploiting Remote Services	Information from Cloud Storage	Ingress Tool Transfer	Exfiltration to Cloud Storage	Data Manipulation
Man-in-the-Middle Attack (MITM)	Account Discovery	Exploiting Valid Accounts	Information from System	Exploitation of Pseudo-Protocol	Weebhook	Data Destruction
Forced Authentication	Permission Groups Discovery	Internal Phishing			Exfiltration Over Alternative Protocol	Resource Hijacking
Exploiting Vulnerabilities	Password Policy Discovery	Alternate Authentication Material Theft				
Credential Dump						
Steal or Forge Credentials and Certificates						
Steal Session Cookies						
Leak Insecurely Stored Credentials						
Input Capture						

5.3. Initial Access

"Initial Access" refers to the technique in which attackers obtain attack entry points through various methods and establish a foothold in the target environment. An attack entry point obtained through an initial compromise could allow the attacker to conduct deeper penetration, such as obtaining valid account information.

Vulnerabilities in Third-party Components or Extensions: Dependencies such as third-party components and libraries introduced by projects and software may contain certain security vulnerabilities. Attackers can gain system permissions, access sensitive data, and destroy system stability by finding and exploiting known vulnerabilities or developing 0-day vulnerabilities. Because the use of these third-party dependencies is so widespread and based on developer trust in popular third-party components, it is often not possible to fully control the security of third-party components or extensions.

5.4. Execution

The "Execution" strategy is the most widely used. Attackers will choose the "Execution" strategy when using malware, APT, etc. to attack. In this tactic, attacker-controlled malicious code is run on a local or remote system, often in combination with other techniques, such as using a remote access tool to execute a PowerShell script.

Package Manager Threats: A package manager is a tool or system used to manage software packages. You can install packages using commands in your package manager. Attackers attack based on the characteristics of the package manager and then use the package manager to execute malicious commands.

5.5. Persistence

"Persistence" is one of the most focused techniques for attackers. Through persistence control, attackers hope to achieve persistent access to the target and minimize the time to access the target. The attacker can maintain an existing connection even if the user reboots or changes credentials; even if the malware on the endpoint is discovered or removed, there is a high probability that it will reappear. In addition to the possibility that the vulnerability has not yet been patched, it could also be that the attacker has already established persistence here or elsewhere on the network.

Backdoor in Code: The attacker inserts malicious code or functionality into the software's code repository to maintain access and control of the system after initial access has been removed or blocked. The attacker can obtain access to a code repository by exploiting a vulnerability or stealing credentials, and then inserting malicious code segments. These malicious code segments are designed to perform specific malicious activities or provide persistent access and control of the system. Attackers can hide within the system and continue performing malicious actions without being detected or blocked.

5.6. Privilege Escalation

"Privilege Escalation" is a technique used by attackers to obtain higher-level privileges on a system or network because, during the intrusion process, some functions and techniques require high privileges. Common methods are to exploit system weaknesses, vulnerabilities, repositories, misconfigurations, etc.

Implant Malicious Dependencies in High-Privilege User Repository: In this technique, the attacker injects malicious code into the dependencies of legitimate open-source libraries or packages in a privileged user repository. In this way, the attacker can gain access to the permissions, privileged resources, or data associated with the privileged user's repository and execute arbitrary code.

5.7. Defense Evasion

"Defense Evasion" refers to the techniques used by attackers to avoid detection by defense systems when attacking targets. Some techniques that can trick antivirus products into

not successfully detecting an intrusion. Techniques an attacker can employ include exploiting vulnerability tool threats, weakening defense mechanisms, or exploiting misconfigurations.

Development Tool Threats: Developer tools are used to perform tasks related to software development, including execution, development, and debugging. These tools can usually be signed using legitimate certificates. Attackers can use compromised trusted tools to evade or bypass security defenses to carry out attacks and hide their activities.

6. Event Verification and Visualization

Under the guidance of the SSC threat portrait proposed above, this paper conducts a series of specific analyses and verifications. The validation answers three main research questions, demonstrating validity and practicality in multiple aspects.

RQ1: Does the above model conform to the characteristics of SSC events? To address this question, this paper applies the proposed SSCM and SSCIM to specific event analysis and demonstrates it through different types of typical events to ensure coverage of diverse attack scenarios.

RQ2: Are the above threat models and indicators recognized? Regarding the question of recognition, this paper uses expert questionnaire evaluation to conduct a multi-dimensional inspection of all aspects of the model and indicators.

RQ3: How to represent arbitrary SSCA events more intuitively based on the above models and indicators? To address this question, this paper uses the analytical results of threat portrait to describe events and presents them in a clear graphical manner to achieve the goal of intuitive representation.

6.1. RQ1: Event Verification

In each event, a comprehensive analysis of key nodes is first performed using the proposed SSCM to verify the different nodes and dependency chains of the supply chain. Secondly, the SSCIM is used to analyze the attack stages and techniques of the incident to verify the different dimensions of the attack threat, so as to comprehensively prove that the proposed portrait model conforms to the characteristics of the supply chain events.

In the dataset construction section, this paper reviews the SSC and collects specific attacks, covering scientific research literature, datasets, news reports, threat reports, vulnerability reports, and analysis blogs. The scientific literature resources are mainly databases such as Google Scholar [52], IEEE [53], ScienceDirect [54]; The main sources of datasets include Backstabber's Knife dataset [19], IQTLabs dataset [55] and CNCF dataset [56]; Sources of texts, including blog posts and reports, comprise online security information websites such as Freebuf [57] and SecWiki [58], dependable vulnerability databases such as CNVD [59] and NVD [60], current reports published by security vendor labs such as Star Map Lab [61] of Qi An Xin, various online cybersecurity discussion forums, blogs, as well as articles on public accounts. Additionally, supplementary information related to the SSC was considered from search engine results. Commonly searched terms include "software supply chain", "supply chain", "software supply chain attack", "supply chain attack", "open-source software supply chain" and "open-source supply chain". For individual events, the name of the event is used as the search term. The dataset records a total of 277 SSCAs from 2003 to 2023, corresponding to a total of 1520 incident texts.

The number of SSCAs in the dataset has shown a growing trend in the past decade. In 2003, only 3 events were recorded, accounting for about 1% of the total, while since 2017, an average of 32 attack events have occurred every year. Among all the collected events, 90 attacks or poisons were aimed at package managers such as npm and PyPI, accounting for about 32.5%. Secondly, the number of attacks on the way to obtain software installation packages reached 89, accounting for about 32.1%. Such attacks usually included intrusion, poisoning, or malicious replacement of installation packages on official websites, application stores, update servers, and other ways to obtain software installation packages. In addition, 81 attacks targeted all aspects of the development process of software and components, accounting for about 29.2%. The frequent attack targets included developers,

development tools, build processes, dependent components, and source code, and there were also malicious developers or maintainers. In addition, 26 events occurred at the level of open-source collaboration, accounting for 9.4%, among which malicious contributions and malicious commits occurred frequently in the open-source community. Finally, there were 24 SSCA events caused by vulnerabilities, accounting for 8.7%.

6.1.1. Codecov Attack Event

The SSCM is used to analyze Codecov events [62]. Codecov's bash uploader script was modified by the attacker, causing users to send sensitive information to a third-party server when using Codecov to upload test data. That is, starting from the core "Software" node, the inner "Code" node is traced outward to the middle "Sofware Source Code", and then the "Sofware Source Code" node is traced outward to the outer "Development Tool" node, and a dependency chain is formed.

By analyzing this event using the SSCIM, the attack chain can be obtained as the Initial Access, Execution, Privilege Escalation, Credential Access, Discovery, Collection and Exfiltration. In the Initial Access, the attacker used the method of "Exploiting Application Vulnerabilities and Errors" to launch intrusion activities through Codecov's incorrect Docker file. During the Execution, the attacker used "Command Exploitation" combined with "IDE Malicious Build Injection" to execute the implanted malicious bash uploader script. In the Privilege Escalation, attacker used "Operational Access Token" combined with malicious scripts to steal customers' tokens, keys, and other credentials, and use these credentials to escalate privilege. During the Credential Access, attackers used "Exposing Insecurely Stored Credentials" and "Stealing or Forging Credentials and Certificates" to steal any credentials, tokens, keys, or authentication certificates passed from the customer's computer. In the Discovery, attacker used "System Information Discovery" to collect "Information From the System" and used "Data from the Code Repository" to obtain environment variables and some configuration information in the script and git remote information in the code repository.

The remaining event verifications selected in this section are shown in Tables 6 and 7 below:

Table 6. SSCM Event Verification.

Event	Dependency Chain
XcodeGhost [49]	Software → Code → Development Tools Software → Installation Package → Resource Sharing,Cloud Service
Event-Streams [50]	1. For EventStreams itself: Software → Code → Developer 2. For affected downstream software: Software → Components → Component Source Code → Third-Party Open-Source Library
PHP Backdoor [63]	Software → Installation Package → Updates by Users → Update Server
Driver Talent [64]	Software → Installation Package → Updates by Users → Update Server
SolarWinds [65]	Software → Installation Package → Official Website
Log4j [10]	For affected downstream software: Software → Components → Component Source Code → Third-Party Open-Source Library

6.1.2. Output Analysis

This paper aims to validate the SSC portrait model using the SSCA event dataset, which comprises 277 occurrences and related 1520 texts. This paper uses statistics and analyzes the output results of the model, focusing on four significant threat surfaces: Installation Package, Code, Runtime Environment, and Components. The model demonstrates a notable level of accuracy on all four threat surfaces, achieving 87.76% and 84.75% accuracy at the component threat surface and code threat surface, respectively. Additionally, the component level exhibits a low false positive rate of 12.24%. This can be attributed to the

availability of a larger number of events and corresponding texts for both types of attacks, as shown in Figure 6.

Table 7. SSCIM Event Verification.

Event	Tactic	Technique
XcodeGhost [49]	Resource Development	Develop Malicious Project
	Initial Access	Phishing
	Execution	API Exploitation, Runtime Backdoor
	Defense Evasion	Subvert Trust Control
	Credential Access	Forced Authentication, MITM, Input Capture
	Collection	Information from System
	Command and Control	Exploitation of Pseudo-Protocol
	Impact	Implant Malicious Module
Event-Streams [50]	Resource Development	Contributing Malicious Code to Open-Source Repository
	Initial Access	Malicious Module Injection
	Execution	User Execution, Runtime Backdoor
	Persistence	Backdoor in Code
	Privilege Escalation	Implant Malicious Dependencies in High-Privilege User Repository
	Credential Access	Leak Insecurely Stored Credentials
PHP Backdoor [63]	Resource Development	Develop Malicious project, Create and Cultivate Account, Hacking the Update Server
	Initial Access	Malicious Module Injection
	Execution	Runtime Backdoor
	Persistence	Backdoor in Code
SolarWinds [65]	Reconnaissance	Collect Target Identity Information
	Resource Development	Malicious Acquisition, Develop Malicious Project
	Initial Access	Exploiting Application Vulnerabilities and Errors, External Remote Services, Exploiting Trusted Third-Party Relationships, Exploiting Valid Accounts
	Execution	Command Execution, IDE Malicious Build Injection
	Persistence	Account Manipulation, External Remote Services, Abuse Task Scheduling
	Defense Evasion	Obfuscate Files or Information,Impair Defense Mechanisms or Components,Indicator Removal
	Credential Access	Leak Insecurely Stored Credentials, Steal or Forge Credentials and Certificates, Credential Dump, Steal Session Cookies
	Discovery	Account Discovery, Permission Groups Discovery, Discovering System Information
	Lateral Movement	Exploiting Remote Services, Alternate Authentication Material theft
	Collection	Information from Code Repository, Information from System
Log4j [10]	Reconnaissance	Active Scanning
	Resource Development	Malicious Acquisition, Develop Malicious Project
	Initial Access	Code Repository Contamination, Development Environment Threats, Vulnerabilities in Third-Party Components/Extensions, Development Tool Threats, Exploiting Application Vulnerabilities and Errors
	Execution	Package Manager Threats, Exploiting Software Vulnerabilities
	Privilege Escalation	Exploiting Software Vulnerabilities
	Defense Evasion	Exploiting Vulnerabilities

Table 7. *Cont.*

Event	Tactic	Technique
Driver Talent [64]	Credential Access	Exploiting Vulnerabilities
	Reconnaissance	Collect Target Identity Information, Collect Target Host Information, Scan Open Information Sources
	Resource Development	Malicious Acquisition, Hacking the Update Server
	Initial Access	Exploiting Valid Accounts
	Execution	User Execution
	Defense Evasion	Exploiting Valid Accounts, Indicator Removal
	Credential Access	Brute Force, Credential Dump
	Lateral Movement	Exploiting Valid Accounts, Exploiting Remote Services
	Collection	Information from System
	Impact	Data Manipulation

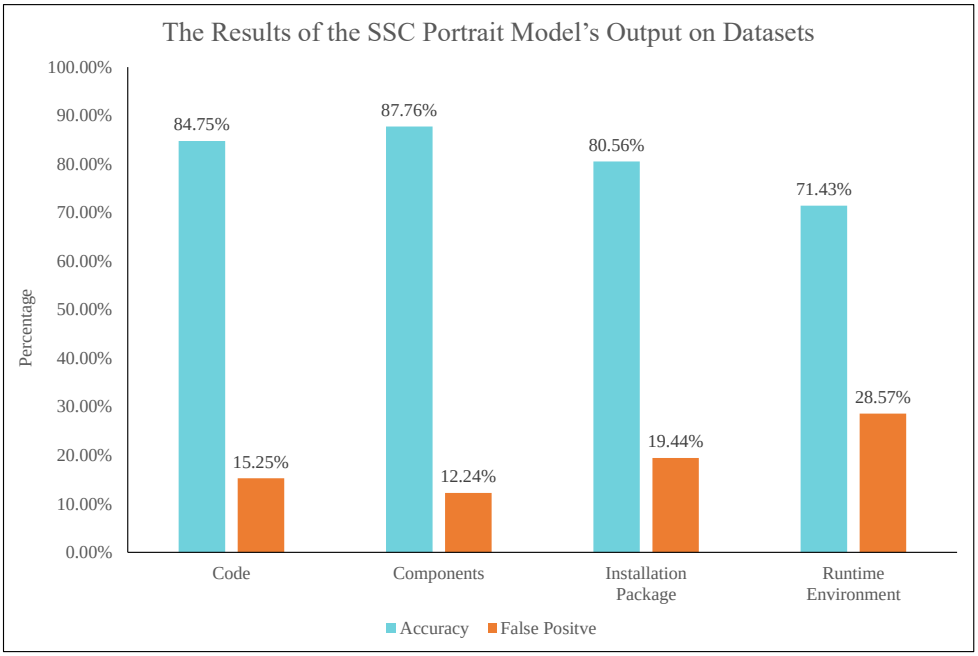

Figure 6. The Results of the SSC Portrait Model's Output on Datasets.

In contrast, the number of attacks on the operational environment threat surface was low. It accounted for only 8.58% of the total number of threats in the entire incident. The component threat surface accounted for the largest percentage, at 36.03%. The Code threat surface and the Installation Package threat surface were second only to the Component's percentage, with the former accounting for a larger percentage, as shown in Figure 7.

This paper focuses on the "Attack Object" part of the model's output and identifies the top 10 attack objects with the highest incidence. These represent the high-frequency attack objects of the SSCAs, as shown in Figure 8. Among the observed attack objects, those identified as "Components" are the most frequent, accounting for 20.75% of the total. These were followed by objects related to "Code Libraries", which constituted 17.86% of the items analyzed. Additionally, objects associated with "Installation Packages" appeared with a relatively high frequency, representing 14.46% of the total.

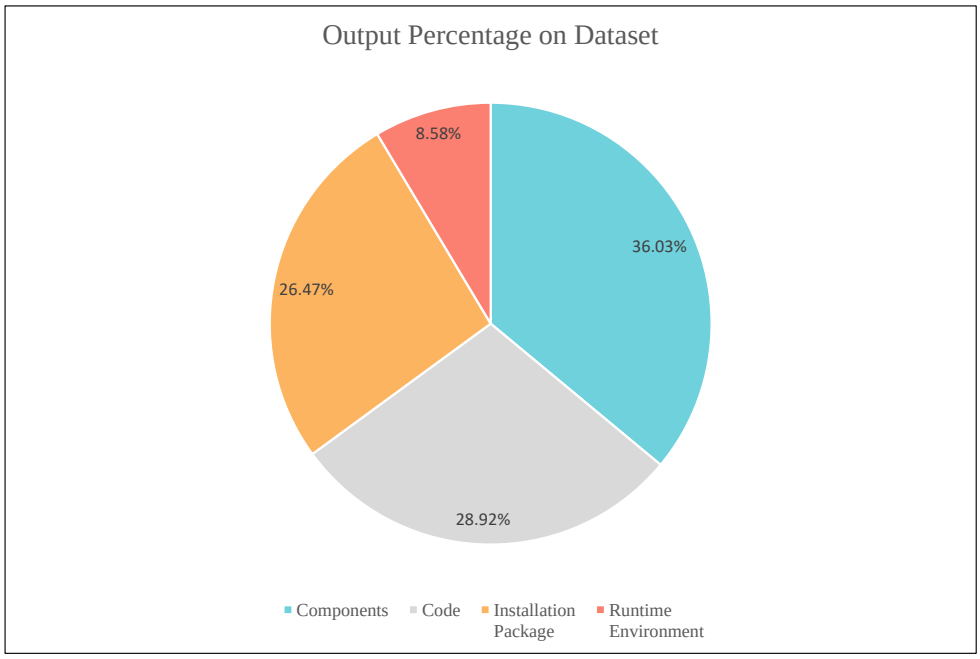

Figure 7. Output percentage on four levels.

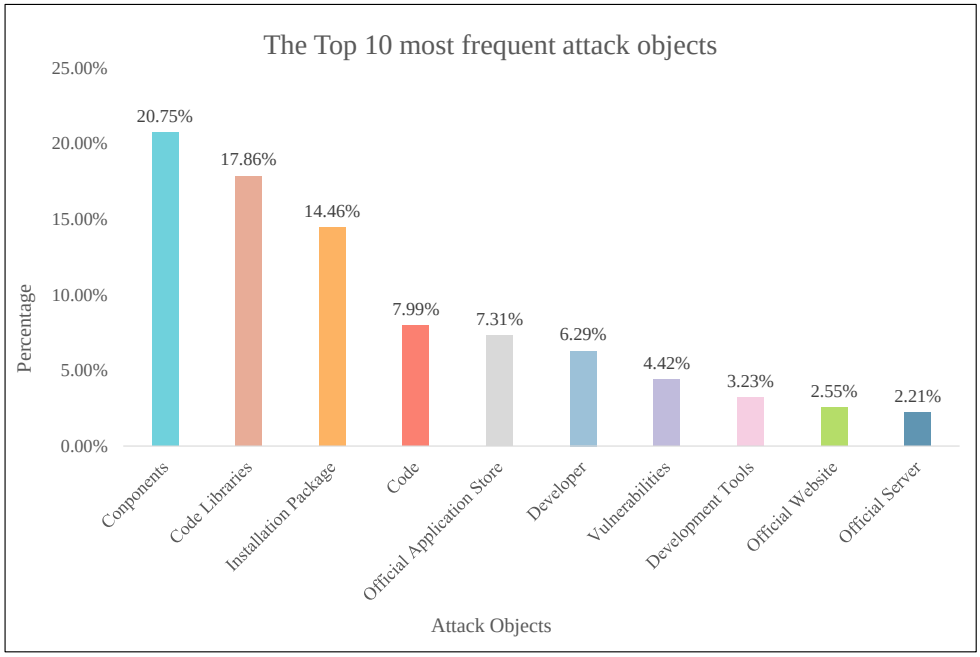

Figure 8. The Top 10 most frequent attack objects.

Overall, our SSC portrait model performs well in a large number of event validations, these remarkable results not only highlight the effectiveness and usefulness of the portrait model but also suggest its potential to effectively, comprehensively, and accurately depict the SSC.

6.2. RQ2: Questionnaire Assessment

The validity of the SSC threat portrait is confirmed through a questionnaire in this paper. The questionnaire assesses eight dimensions of evaluation for the SSCM and SSCIM, including understandability, structural acceptance, correctness, name acceptance, position acceptance, comprehensiveness, effectiveness, and usefulness. The purpose is to verify the validity of the SSC threat portrait from different perspectives, as shown in Table 8.

Among the 40 questionnaires, more than 72.5% of respondents agreed with all dimensions of the model and indicator matrix, and the average score of each dimension exceeded 7.5 points (out of 10 points), as shown in Table 9. Over 77.5% of respondents agreed that the SSCM and SSCIM were easy to understand, with an average rating of at least 7.9. Additionally, more than 80% of respondents had a positive attitude toward structural acceptance, with an average rating of more than 7.75. Over 82.5% of respondents agreed that the nodes were accurate at all levels, with ratings centered around 7 or 8. The comprehensive assessment of all nodes about their naming and categorization yielded favorable results, with an approval rate of more than 80%. With over 72.5% of the participants, the majority agreed with the comprehensiveness. Meanwhile, an examination of the questionnaires revealed the need to augment the degree of comprehensive coverage to encompass a wider range of elements. In regards to effectiveness and utility, participants rated the model and matrix as useful resources for comprehending, proficiently analyzing, and protecting the SSC's attack surface. This evaluation was endorsed by more than 82.5% of respondents, with an average score exceeding 8 points. This confirms the SSC threat portrait model's credibility to a certain extent.

Table 8. SSC Threat Portrait Evaluation Perspective.

Assessment Dimensions	Sample Questions of SSCM	Sample Questions of SSCIM
Understandability	Is it easy to understand the various levels and nodes of the assessment model?	How easily understandable is the content of this technical matrix?
Naming Acceptance	How well do you agree with the names of different levels and nodes of the model?	How do you agree with the naming of different tactics and techniques?
Structural Acceptance	What are your thoughts on the overall structure of this model?	What are your thoughts on the overall structure of this technology matrix?
Position Acceptance	Evaluate the accuracy of each node position?	Evaluate the accuracy of the tactical phases to which techniques are assigned in the technique matrix?
Correctness	Evaluate the correctness of node settings at each layer of the model?	How accurate are the sub-techniques within the matrix?
Comprehensiveness	How comprehensive is the model's coverage of attack points in the SSCAs?	Evaluate this technique matrix's coverage of SSCA techniques?
Effectiveness	How effective is the node chain formed by the model for analyzing a specific SSCA event?	Evaluate whether this matrix effectively analyzes the techniques used by attackers for a specific SSC incident?
Usefulness	Is this model useful for understanding, analyzing, and defending against the attack surface of the SSC?	Evaluate the usefulness of this matrix for analyzing and defending against SSCAs?

6.3. RQ3: Visualization

- Visualization of Event-Streams Events

Event-Stream events [50] mainly exploit people's trust in open-source community code maintainers, as shown in Figure 9. First, for the Event-Stream source code itself, the threat penetration point is traced from the internal "Code" node to the external "Developer" node. Then move to the "Third-Party Open-Source Library" at the same level through the "Publish Third-Party Components" node. Second, for the affected downstream code, the threat can be traced from the inner "Code" node to the middle "Third-Party Component" node, and the "Third-Party Component" node can be traced to the "Third-Party Open-Source Library" node through the "Introduction of Third-Party Components From the Open-Source Market or Commercial Library".

Table 9. Questionnaire Data Statistics.

Dimensions	SSCM			SSCIM		
	Mean	Median	Acceptance Percentage *	Mean	Median	Acceptance Percentage *
Understandability	8.03	8.00	77.50%	7.90	8.00	80.00%
Naming Acceptance	7.93	8.00	80.00%	7.85	8.00	85.00%
Structural Acceptance	7.75	8.00	80.00%	7.98	8.00	82.50%
Position Acceptance	7.83	8.00	85.00%	8.13	8.00	87.50%
Correctness	7.90	8.00	82.50%	7.93	8.00	87.50%
Comprehensiveness	7.50	7.00	72.50%	7.58	7.50	77.50%
Effectiveness	8.28	8.00	85.00%	8.03	8.00	82.50%
Usefulness	8.20	9.00	85.00%	8.25	9.00	85.00%

* A numerical value equal to or exceeding 7 indicates a heightened level of recognition of the specific dimension as perceived by the survey participants.

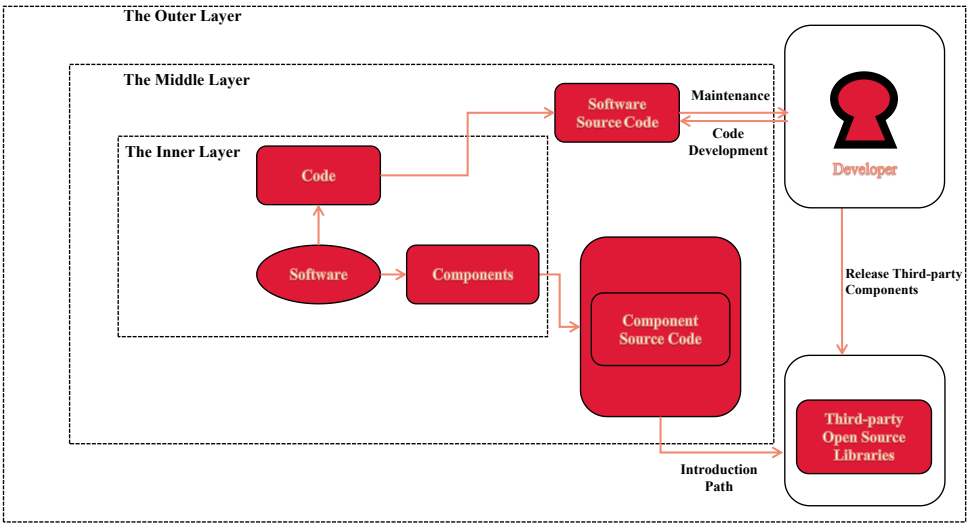

Figure 9. Visualization of EventStream event model analysis.

Analyze the techniques and tactics that may be included in Event-Stream events based on the SSCIM. The attack chain is shown in Figure 10, which is Resource Development, Initial Access, Execution, Persistence, Privilege Escalation, and Credential Access.

- Visualization of XcodeGhost attack event

The XcodeGhost attack [49] occurred at the software code level in the SSCM. As shown in Figure 11, the penetration point is traced from the internal "Code" node to the external "Development Tool" node, that is, the attacker used the malicious attack on the upstream development tool Xcode to cause the risk of all downstream software developed with it. In terms of transmission channels, the developer in this incident chose to obtain the Xcode installation package from unofficial channels such as cloud disk links, cloud services, and other resource sharing; that is, the penetration point can be traced from the "Installation Package" node inside the SSCM to the external "Resource Sharing" and "Cloud Service" nodes.

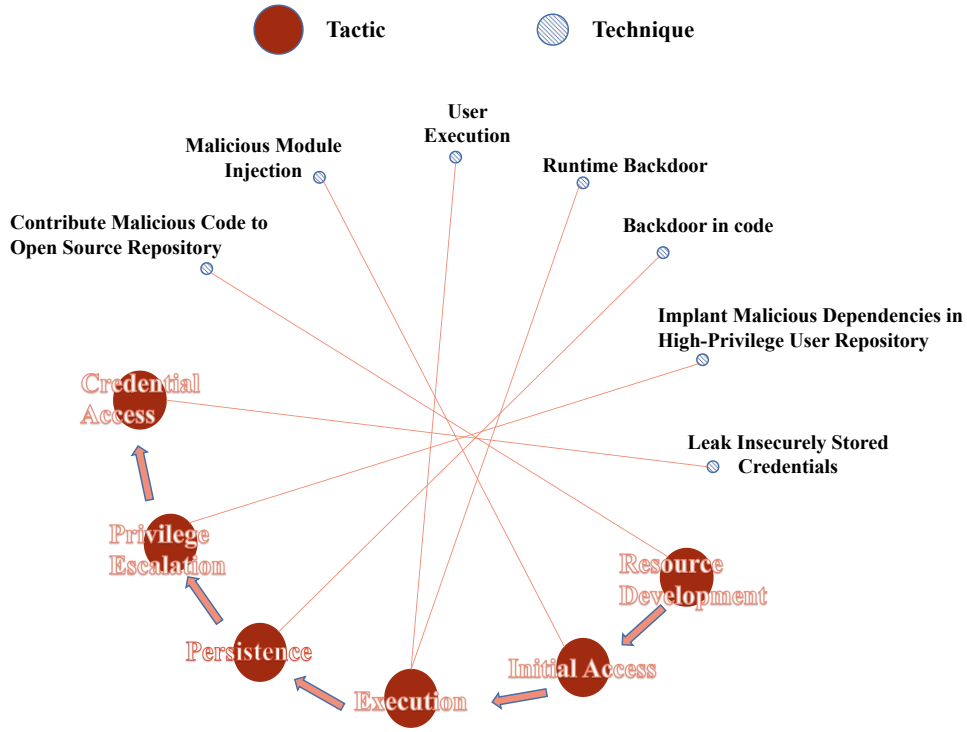

Figure 10. Visualization of EventStream attack indicator analysis.

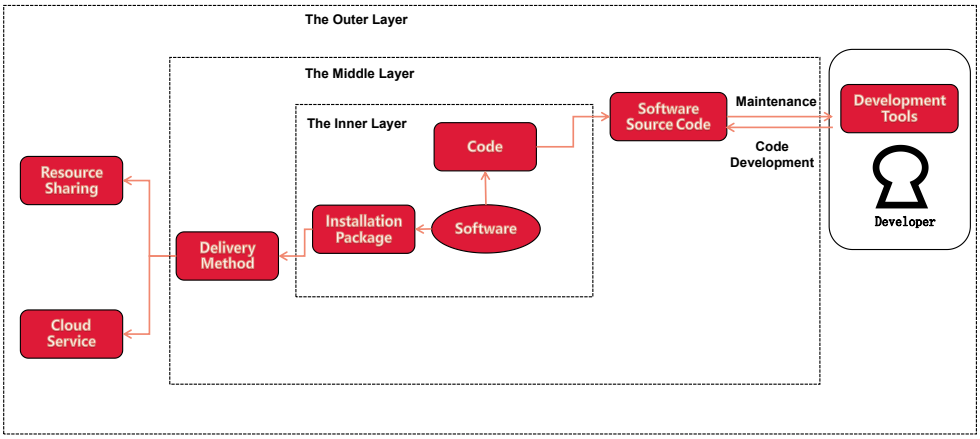

Figure 11. Visualization of XcodeGhost model analysis.

The attacker's attack chain in the SSCIM is resource development, initial intrusion, execution, persistent control, defense evasion, obtaining legal credential access, collection, command control, and harm, as shown in Figure 12.

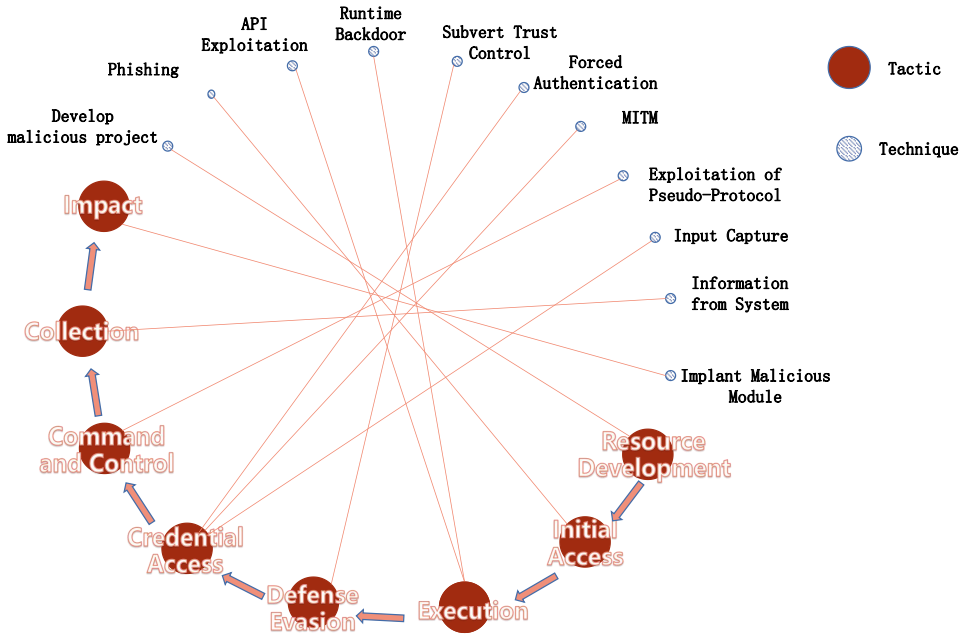

Figure 12. Visualization of XcodeGhost attack indicator analysis.

7. Conclusions

The SSC has become increasingly complex with the development of technology, and it is urgent to have an in-depth understanding of the SSC and attackers' attack methods. In the face of increasingly complex upstream and downstream dependencies in the supply chain, the traditional development, distribution, and usage models appear to be rough in describing the supply chain, and the portrait needs to be improved. Based on the existing SSC models and events, this paper uses the LLM ChatGPT [7] and artificial experience to mine the common characteristics of the supply chain from event analysis, and proposes a chain perspective model that includes more key links. Based on Attack Net [8] and the framework of MITRE ATT&CK [9], the attack indicator matrix is further constructed to describe the techniques and tactics used by attackers in the entire SSC attack chain. At the same time, the effectiveness of the proposed threat portrait is verified and visualized through a large number of events and expert experience evaluation. The complete indicator matrix has been open-sourced on Github [66]. The above work helps to understand the possible supply chain attack risk and determine the defense strategy.

Author Contributions: Conceptualization, all authors; methodology, M.W.; validation, M.W.; investigation, M.W.; resources, P.W. and Q.L.; writing—original draft preparation, M.W.; writing—review and editing, M.W.; visualization, M.W.; supervision, P.W. and Q.L.; project administration, P.W. and Q.L.; funding acquisition, Q.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China under Grant 61902328; and in part by the Key R&D projects of Sichuan Science and technology plan (2022YFG0323); and in part by the Key R&D projects of Chengdu Science and technology plan (2022-YF05-00451-SN).

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Peisert, S.; Schneier, B.; Okhravi, H.; Massacci, F.; Benzel, T.; Landwehr, C.; Mannan, M.; Mirkovic, J.; Prakash, A.; Michael, J.B. Perspectives on the SolarWinds incident. *IEEE Secur. Priv.* **2021**, *19*, 7–13. [CrossRef]
2. In-Depth Aanalysis of the Supply Chain Attack Case of CCleaner Backdoor Code-Compilation Environment Pollution. Available online: https://ti.qianxin.com/blog/articles/in-depth-analysis-of-ccleaner-malware/ (accessed on 15 March 2023).
3. The State of Software Supply Chain Security. Available online: https://www.reversinglabs.com/resources/the-state-of-software-supply-chain-security (accessed on 15 March 2023).
4. How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks. Available online: https://www.gartner.com/en/documents/4003625 (accessed on 1 March 2023).
5. Zhenfei, Z. Research on Pollution Mechanism and Defense of Software Supply Chain. Master's Thesis, Beijing University of Posts and Telecommunications, Bejing, China, 2018.
6. Du, S.; Lu, T.; Zhao, L.; Xu, B.; Guo, X.; Yang, H. Towards an analysis of software supply chain risk management. In Proceedings of the World Congress on Engineering and Computer Science, San Francisco, CA, USA, 23–25 October 2013; Volume 1.
7. Introducing ChatGPT. Available online: https://openai.com/blog/chatgpt (accessed on 1 May 2023).
8. Steffan, J.; Schumacher, M. Collaborative attack modeling. In Proceedings of the 2002 ACM Symposium on Applied Computing, Madrid, Spain, 11–14 March 2002; pp. 253–259.
9. ATT&CK Matrix. Available online: https://attack.mitre.org (accessed on 1 March 2023).
10. Technical Advisory: Zero-Day Critical vulnerability in Log4j2 Exploited in the Wild. Available online: https://www.bitdefender.com/blog/businessinsights/technical-advisory-zero-day-critical-vulnerability-in-log4j2-exploited-in-the-wild/ (accessed on 1 March 2023).
11. Torres-Arias, S.; Afzali, H.; Kuppusamy, T.K.; Curtmola, R.; Cappos, J. in-toto: Providing farm-to-table guarantees for bits and bytes. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1393–1410.
12. Software Supply Chain Attacks. Available online: https://www.whitesourcesoftware.com/resources/blog/software-supply-chain-attacks/ (accessed on 1 March 2023).
13. Ji, S.; Wang, Q.; Chen, A.; Zhao, B.; Ye, T.; Zhang, X.; Wu, J.; Li, Y.; Yin, J.; Wu, T. Review of open source software supply chain security research. *J. Softw.* **2022**, *34*, 1330–1364.
14. Benthall, S. Assessing software supply chain risk using public data. In Proceedings of the 2017 IEEE 28th Annual Software Technology Conference (STC), Gaithersburg, MD, USA, 25–28 September 2017; pp. 1–5.
15. Pfretzschner, B.; ben Othmane, L. Identification of dependency-based attacks on node.js. In Proceedings of the 12th International Conference on Availability, Reliability and Security, Reggio Calabria, Italy, 29 August–1 September 2017; pp. 1–6.
16. Gokkaya, B.; Aniello, L.; Halak, B. Software supply chain: Review of attacks, risk assessment strategies and security controls. *arXiv* **2023**, arXiv:2305.14157.
17. Liu, C.; Chen, S.; Fan, L.; Chen, B.; Liu, Y.; Peng, X. Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 21–29 May 2022; pp. 672–684.
18. Zimmermann, M.; Staicu, C.A.; Tenny, C.; Pradel, M. Small world with high risks: A study of security threats in the npm ecosystem. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 995–1010.
19. Ohm, M.; Plate, H.; Sykosch, A.; Meier, M. Backstabber's knife collection: A review of open source software supply chain attacks. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, (Proceedings 17), Lisbon, Portugal, 24–26 June 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 23–43.
20. Zahan, N.; Zimmermann, T.; Godefroid, P.; Murphy, B.; Maddila, C.; Williams, L. What are weak links in the npm supply chain? In Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, Pittsburgh, PA, USA, 25–27 May 2022; pp. 331–340.
21. Dey, T.; Mockus, A. Are software dependency supply chain metrics useful in predicting change of popularity of npm packages? In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, Oulu, Finland, 10 October 2018; pp. 66–69.
22. Gonzalez, D.; Zimmermann, T.; Godefroid, P.; Schäfer, M. Anomalicious: Automated detection of anomalous and potentially malicious commits on github. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Madrid, Spain, 25–28 May 2021; pp. 258–267.
23. Duan, R.; Alrawi, O.; Kasturi, R.P.; Elder, R.; Saltaformaggio, B.; Lee, W. Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv* **2020**, arXiv:2002.01139.
24. Tang, W.; Luo, P.; Fu, J.; Zhang, D. Libdx: A cross-platform and accurate system to detect third-party libraries in binary code. In Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 14–21 February 2020; pp. 104–115.
25. Ladisa, P.; Plate, H.; Martinez, M.; Barais, O. Sok: Taxonomy of attacks on open-source software supply chains. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 1509–1526.
26. Bos, A.M. A Review of Attacks Against Language-Based Package Managers. *arXiv* **2023**, arXiv:2302.08959.

27. Reed, M.; Miller, J.F.; Popick, P. *Supply Chain Attack Patterns: Framework and Catalog*; Office of the Deputy Assistant Secretary of Defense for Systems Engineering: Washington, DC, USA, 2014; Volume 2.
28. Supply Chain Attack Framework and Attack Patterns. Available online: https://www.mitre.org/sites/default/files/publications/supply-chain-attack-framework-14-0228.pdf (accessed on 1 March 2023).
29. Buchicchio, E.; Grilli, L.; Capobianco, E.; Cipriano, S.; Antonini, D. Invisible supply chain attacks based on trojan source. *Computer* **2022**, *55*, 18–25. [CrossRef]
30. Neil, L.; Mittal, S.; Joshi, A. Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports. In Proceedings of the 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), Miami, FL, USA, 9–11 November 2018; pp. 7–12.
31. Neupane, S.; Holmes, G.; Wyss, E.; Davidson, D.; De Carli, L. Beyond Typosquatting: An In-depth Look at Package Confusion. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 3439–3456.
32. Zahan, N. Software Supply Chain Risk Assessment Framework. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Melbourne, Australia, 14–20 May 2023; pp. 251–255.
33. Ohm, M.; Sykosch, A.; Meier, M. Towards detection of software supply chain attacks by forensic artifacts. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Virtual, 25–28 August 2020; pp. 1–6.
34. Zhenhua, W. Research on Pollution Detection Technology of Software Supply Chain. Master's Thesis, The Information Engineering University, Henan, China, 2019.
35. Vu, D.L.; Pashchenko, I.; Massacci, F.; Plate, H.; Sabetta, A. Towards using source code repositories to identify software supply chain attacks. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 2093–2095.
36. Wang, X. On the feasibility of detecting software supply chain attacks. In Proceedings of the MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM), San Diego, CA, USA, 29 November 2021–2 December 2021; pp. 458–463.
37. Purba, M.D.; Chu, B. Extracting Actionable Cyber Threat Intelligence from Twitter Stream. In Proceedings of the 2023 IEEE International Conference on Intelligence and Security Informatics (ISI), Charlotte, NC, USA, 2–3 October 2023; pp. 1–6.
38. Wang, P.; Dai, G.; Zhai, L. Event-Based Threat Intelligence Ontology Model. In Proceedings of the International Conference on Science of Cyber Security, Shanghai, China, 13–15 August 2023; pp. 261–282.
39. Perrina, F.; Marchiori, F.; Conti, M.; Verde, N.V. AGIR: Automating Cyber Threat Intelligence Reporting with Natural Language Generation. *arXiv* **2023**, arXiv:2310.02655.
40. Fayyazi, R.; Yang, S.J. On the Uses of Large Language Models to Interpret Ambiguous Cyberattack Descriptions. *arXiv* **2023**, arXiv:2306.14062.
41. Ali, T.; Kostakos, P. HuntGPT: Integrating Machine Learning-Based Anomaly Detection and Explainable AI with Large Language Models (LLMs). *arXiv* **2023**, arXiv:2309.16021.
42. Sun, Y.; Wu, D.; Xue, Y.; Liu, H.; Wang, H.; Xu, Z.; Xie, X.; Liu, Y. When GPT Meets Program Analysis: Towards Intelligent Detection of Smart Contract Logic Vulnerabilities in GPTScan. *arXiv* **2023**, arXiv:2308.03314.
43. Wang, Z.; Zhang, L.; Cao, C.; Liu, P. The Effectiveness of Large Language Models (Chatgpt and Codebert) for Security-Oriented Code Analysis. SSRN 2023, SSRN:4567887. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4567887 (accessed on 7 November 2023).
44. Setianto, F.; Tsani, E.; Sadiq, F.; Domalis, G.; Tsakalidis, D.; Kostakos, P. GPT-2C: A parser for honeypot logs using large pre-trained language models. In Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Virtual, 8–11 November 2021; pp. 649–653.
45. Le, V.H.; Zhang, H. Log Parsing with Prompt-based Few-shot Learning. *arXiv* **2023**, arXiv:2302.07435.
46. Ranade, P.; Piplai, A.; Joshi, A.; Finin, T. Cybert: Contextualized embeddings for the cybersecurity domain. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; pp. 3334–3342.
47. He, J.; Wang, L.; Hu, Y.; Liu, N.; Liu, H.; Xu, X.; Shen, H.T. ICL-D3IE: In-context learning with diverse demonstrations updating for document information extraction. *arXiv* **2023**, arXiv:2303.05063.
48. Wei, X.; Cui, X.; Cheng, N.; Wang, X.; Zhang, X.; Huang, S.; Xie, P.; Xu, J.; Chen, Y.; Zhang, M.; et al. Zero-shot information extraction via chatting with chatgpt. *arXiv* **2023**, arXiv:2302.10205.
49. XcodeGhost. Available online: https://en.wikipedia.org/w/index.php?title=XcodeGhost&oldid=1022461786 (accessed on 15 March 2023).
50. I Don't Know What to Say. Available online: https://github.com/dominictarr/event-stream/issues/116 (accessed on 1 March 2023).
51. A New Open Framework For Releasing Secure Products. Available online: https://pbom.dev/#overview (accessed on 1 May 2023).
52. Goggle Scolar. Available online: https://scholar.google.com/ (accessed on 2 March 2023).
53. IEEE. Available online: https://ieeexplore.ieee.org/ (accessed on 2 March 2023).
54. Sciencedirect. Available online: https://www.sciencedirect.com (accessed on 2 March 2023).
55. Software Supply Chain Compromises. Available online: https://github.com/in-toto/supply-chain-compromises (accessed on 15 March 2023).

56. Catalog of Supply Chain Compromises. Available online: https://github.com/cncf/tag-security/tree/main/supply-chain-security (accessed on 15 March 2023).
57. FreeBuf. Available online: https://www.freebuf.com (accessed on 15 March 2023).
58. SecWiki. Available online: https://secwiki.org/w/Main_Page (accessed on 15 March 2023).
59. CNVD. Available online: https://www.cnvd.org.cn (accessed on 15 March 2023).
60. NVD. Available online: https://nvd.nist.gov (accessed on 15 March 2023).
61. Star Map Lab. Available online: https://tianwen.qianxin.com/blog/ (accessed on 15 March 2023).
62. Post-Mortem/Root Cause Analysis. Available online: https://about.codecov.io/apr-2021-post-mortem/ (accessed on 2 March 2023).
63. Hackers Backdoor PHP Source Code after Breaching Internal Git Server. Available online: https://arstechnica.com/gadgets/2021/03/hackers-backdoor-php-source-code-after-breaching-internal-git-server/ (accessed on 15 April 2023).
64. "Driver Talent" Trojan Detailed Analysis Report Infected 100,000 Computers to Mine Monero in 2 h. Available online: https://s.tencent.com/research/report/610.html (accessed on 15 March 2023).
65. Martínez, J.; Durán, J.M. Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study. *Int. J. Saf. Secur. Eng.* **2021**, *11*, 537–545. [CrossRef]
66. Supply-Chain-Attack. Available online: https://github.com/kcrio/supply-chain-attack (accessed on 30 August 2023).