# SoK: Taxonomy of Attacks on Open-Source Software Supply Chains

Piergiorgio Ladisa*‡, Henrik Plate*, Matias Martinez†, and Olivier Barais‡,
*SAP Security Research †Université Polytechnique Hauts-de-France ‡Université de Rennes 1, Inria, IRISA
{piergiorgio.ladisa, henrik.plate}@sap.com, matias.martinez@uphf.fr, {piergiorgio.ladisa, olivier.barais}@irisa.fr

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179304

***Abstract*—The widespread dependency on open-source software makes it a fruitful target for malicious actors, as demonstrated by recurring attacks. The complexity of today's open-source supply chains results in a significant attack surface, giving attackers numerous opportunities to reach the goal of injecting malicious code into open-source artifacts that is then downloaded and executed by victims.**

**This work proposes a general taxonomy for attacks on open-source supply chains, independent of specific programming languages or ecosystems, and covering all supply chain stages from code contributions to package distribution. Taking the form of an attack tree, it covers 107 unique vectors, linked to 94 real-world incidents, and mapped to 33 mitigating safeguards.**

**User surveys conducted with 17 domain experts and 134 software developers positively validated the correctness, comprehensiveness and comprehensibility of the taxonomy, as well as its suitability for various use-cases. Survey participants also assessed the utility and costs of the identified safeguards, and whether they are used.**

***Index Terms*—Open Source, Security, Software Supply Chain, Malware, Attack**

## I. Introduction

*Software supply chain attacks* aim at injecting malicious code into software components to compromise downstream users. Recent incidents, like the infection of SolarWind's Orion platform [1], downloaded by approx. 18,000 customers, including government agencies and providers of critical infrastructure, demonstrate the reach and potential impact of such attacks. Accordingly, software supply chain attacks are among the primary threats in today's threat landscape, as reported by ENISA [2] or the *US Executive Order on Improving the Nation's Cybersecurity* [3].

This work focuses on the specific instance of attacks on Open-Source Software (OSS) supply chains, which exploit the widespread use of open-source during the software development lifecycle as a means for spreading malware. Considering the dependency of the software industry on open-source – across the technology stack and throughout the development lifecycle, from libraries and frameworks to development, test and build tools, Ken Thompson's reflections [4] on trust (in code and its authors) is more relevant than ever. Indeed, attackers abuse trust relationships existing between the different open-source stakeholders [5], [6]. The appearance and significant increase of attacks on OSS throughout the last few years, as reported by Sonatype in their 2021 report [7], demonstrate that attackers consider them a viable means for spreading malware.

Recently, industry and government agencies increased their efforts to improve software supply chain security, both in general and in regards to open-source. MITRE, for instance, proposes an end-to-end framework to preserve supply chain integrity [8], and the OpenSSF develops the SLSA framework, which groups several security best-practices for open-source projects [9]. Academia contributes an increasing number of scientific publications, many of which get broad attention in the developer community, e.g., [10] or [11].

Nevertheless, we observed that existing works on open-source supply chain security lack a comprehensive, comprehensible, and general description of how attackers inject malicious code into OSS projects, that is independent of specific programming languages, ecosystems, technologies, and stakeholders.

We believe a taxonomy classifying such attacks could be of value for both academia and industry. Serving as a common reference and clarifying terminology, it could support several activities, e.g., developer training, risk assessment, or the development of new safeguards. As such, we set out to answer the following research questions:

**RQ1 – Taxonomy of attacks on OSS supply chains**

- RQ1.1 – What is a comprehensive list of general attack vectors on OSS supply chains?
- RQ1.2 – How to represent those attack vectors in a comprehensible and useful fashion?

**RQ2 – Safeguards against OSS supply chain attacks**

- RQ2.1 – Which general safeguards exist, and which attack vectors do they address?
- RQ2.2 – What is the utility and cost of those safeguards?
- RQ2.3 – Which safeguards are used by developers?

To answer those questions, we first study both the scientific and grey literature to compile an extensive list of attack vectors, including ones that have been exploited, but also non-exploited vulnerabilities and plausible proofs-of-concept. We then outline a taxonomy in the form of an attack tree. From the identified attacks, we list the associated safeguards. Finally, we conduct two user surveys aiming to validate the attack taxonomy and to collect qualitative feedback regarding the utility, costs, awareness, and use of safeguards.

To this extent, the main contributions of our work are as follows:

- A taxonomy of **107 unique attack vectors** related to OSS supply chains, taking the form of an attack tree and **validated by 17 domain experts** in terms of complete-


DOI 10.1109/SP46215.2023.00010

ness, comprehensibility, and applicability in different use cases.
- A set of **33 safeguards** geared towards the proposed taxonomy, and qualitatively **assessed regarding utility and costs** by the same 17 domain experts.
- The qualitative assessment of **134 developers** on the awareness of selected high-level attack vectors and the corresponding level of protection.

Using an interactive visualization of the attack tree, the taxonomy with descriptions, examples of real-world attacks, references, and associated safeguards can be explored online[1].

The remainder of the paper is organized as follows. Section II introduces basic concepts and elements of OSS supply chains, the assumed attacker model, and the concept of attack trees. Section III describes the methodology applied, comprising the three steps Systematic Literature Review (SLR), modeling of taxonomy and safeguards, and survey design. Section IV details the proposed taxonomy and presents the results of the expert and developer validation. Section V introduces the safeguards associated with the aforementioned attack vectors and presents both the experts' feedback on their utility and costs, and the developers' feedback on awareness and use. Section VI discusses the differences between programming languages and highlights the benefits of our work on research. Section VII provides demographic information about the survey participants. Section VIII mentions related works, and Section IX discusses threats to the validity of our work. Finally, the conclusion and outlook are provided in Section X.

## II. Background

This section describes, at a high level, the systems and stakeholders involved in the development, build, and distribution of OSS artifacts (cf. Figure 1). They are constituting elements of OSS supply chains and contribute to their attack surface. They commonly interact in a distributed setting [12], even if the specifics differ from one OSS project to another.

The section concludes with a description of the attacker model considered throughout the paper, and a summary of the concept of attack trees.

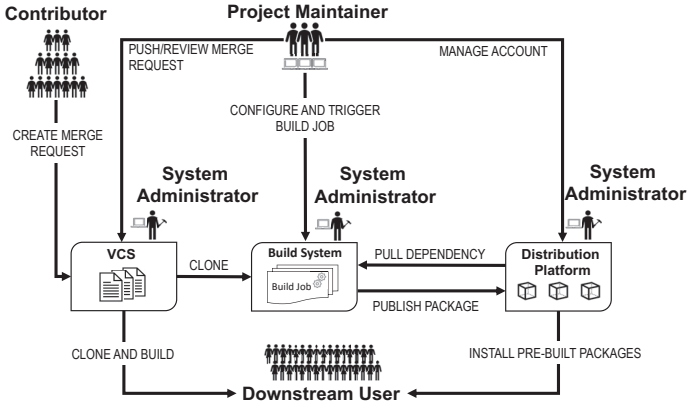


Fig. 1: Stakeholders, systems and dataflows related to the development, build and distribution of OSS artifacts.

[1]https://sap.github.io/risk-explorer-for-software-supply-chains/

### A. Systems

The systems considered comprise Version Control System (VCS), build systems, and package repositories. They do not necessarily correspond to concrete physical or virtual systems providing the respective function but should be seen as roles, multiple of which can be exercised by a single host or 3rd party service.

**Version Control Systems** host the source code of the OSS project, not only program code but also metadata, build configuration, and other resources. They track and manage all the changes of the codebase that happen throughout the development process. Plain VCSs like Git do not require its users to authenticate, but complementary tools and 3rd party services offer additional functionalities (e.g., issue trackers) or security controls (e.g., authentication, fine-grained permissions or review workflows).

**Build Systems** take a project's codebase as input and produce a binary artifact, e.g., an executable or compressed archive, which can be distributed to downstream users for easy consumption. The build commonly involves so-called dependency or package managers [13], [14], e.g., pip for Python, which determine and download all dependencies necessary for the build to succeed, e.g., test frameworks or OSS libraries integrated into the project at hand. Continuous Integration (CI)/Continuous Delivery (CD) pipelines, running on build automation tools like Jenkins, automate the test, build, and deployment of project artifacts.

**Distribution Platforms** distribute pre-built OSS artifacts to downstream users, e.g., upon the execution of package managers or through manual download. Our definition does not only cover well-known public package repositories like PyPI or Maven Central but also internal and external mirrors, Content Delivery Network (CDN) or proxies.

**Workstations of OSS Maintainers and Administrators.** OSS project maintainers and administrators of the abovementioned systems have privileged access to sensitive resources, e.g., the codebase, a build system's web interface, or a package repository's database. Therefore, their workstations are in the scope of the attack scenario.

### B. Stakeholders

The stakeholders considered comprise OSS project maintainers, contributors, and consumers – as well as administrators of various systems or services involved. Again, they should be understood as roles [15], multiple of which can be assumed by a given individual. For example, maintainers of an OSS project typically consume artifacts of other projects.

**Contributors** contribute code to an OSS project, with limited (read-only) access to project resources. They typically submit contributions to the VCS via merge requests, which are reviewed by project maintainers prior to being integrated.

**OSS Project Maintainers** have privileged access to project resources, e.g., to review and integrate contributors' merge requests, configure build systems and trigger build jobs, or deploy ready-made artifacts on package repositories. The real names of project collaborators, both contributors and maintainers, are not necessarily known. Accounts, including

anonymous ones, gain trust through continued contributions of quality, thanks to which they may be promoted to maintainers (known as a meritocracy).

**System and Service Administrators** have the responsibility to configure, maintain, and operate any of the abovementioned systems or services, e.g., employees of 3rd-party Git hosting providers, members of OSS foundations that operate own build systems for their projects, or employees of companies running package repositories like npm.

**Downstream Users** consume OSS project artifacts, e.g., its source code from the project's VCSs (e.g., through cloning), or pre-built packages from distribution platforms. In the context of downstream development projects, the download is typically automated by package managers like pip or npm, which identify and obtain dozens or hundreds [16], [17] of the project's direct and transitive dependencies.

### C. Risks of Open-Source Software Supply Chains

OSS is widely used by organizations and individuals across the technology stack and throughout the software development lifecycle. Package managers automate its download and installation to a great extent, e.g., when resolving transitive dependencies or updating versions.

The above-described systems are inherently distributed, and the stakeholders are partly unknown or anonymous. They exist for every single open-source component used, which multiplies an attack surface having both technical and social facets. Moreover, even heavily used open-source projects receive only little funding and contributions [18], making it difficult for maintainers to securely run projects and increasing their susceptibility to social-engineering attacks, e.g., when reviewing contributions.

Downstream consumers have no control over and limited visibility into given projects' security practices. The sheer number of dependencies [16] makes rigorous reviews impractical for a given consumer, forcing them to trust the community for a timely detection of vulnerabilities and attacks.

Attackers' primary objectives are data exfiltration, droppers, denial of service, or financial gain [19]. Hence, the larger the user base, direct and indirect, the more attractive an open-source project becomes for attackers. As in other adversarial contexts, attackers require finding single weaknesses, while defenders needs to cover the whole attack surface, which in this case spans the whole supply chain.

### D. Attack Tree

Attack trees [20], [21] are intuitive and systematic representations of attacker goals and techniques, and support organizations in risk assessment, esp. with regards to understanding exposure and identifying countermeasures.

The root node of an attack tree represents the attacker's top-level goal, which is iteratively refined by its children into subgoals. Depending on the degree of refinement, the leaves correspond to more or less concrete and actionable tasks.

As taxonomies require assigning instances to exactly one class, we only consider disjunctive refinement, where child nodes represent alternatives to reach the parent goal.

### E. Attacker Model

The development of the taxonomy was based on the following assumptions and attacker model.

The attacker's top-level goal is to place malicious code in open-source artifacts such that it is executed in the context of downstream projects, e.g., during its development or runtime. Such malware can exfiltrate data, represent or open a backdoor, as well as download and execute second-stage payload (e.g., cryptominers [19]). Targeted assets can belong both to developers of downstream software projects, or their end-users, depending on the attacker's specific intention. However, the focus of the taxonomy is not on *what* malicious code does, but *how* attackers place it in upstream projects.

Insider attacks are out of scope, i.e., adversaries are neither maintainers of the attacked open-source project nor members or employees of 3rd party service providers involved in the development, build, or distribution of project artifacts. As such, attackers do not have any privileged access to project resources like build jobs or infrastructure like the server or database underlying code repositories.

Initially, they only have access to publicly available information and publicly accessible resources, which they can collect and analyze following the Open Source Intelligence (OSINT) [22] approach. Of course, due to the nature of open-source projects, many project details are freely accessible, e.g., project dependencies, build information, or commit and merge request histories. Attackers can interact with any of the stakeholders and resources depicted in Figure 1, e.g., to communicate with maintainers using merge requests or issue trackers or to create fake accounts and projects.

## III. Methodology

The methodology adopted to answer the above-mentioned research questions comprises three phases (cf. Figure 2).

First, we review scientific and grey literature to collect an extensive list of attack vectors on OSS supply chains.

Second, starting from the vectors described in the literature and the OSS supply chain elements introduced in Section II, we abstract from specific programming languages or ecosystems, perform threat modeling, and create a taxonomy that takes the form of an attack tree. Also, we identify and classify safeguards mitigating those vectors.

Third, to validate the proposed taxonomy and the list of safeguards, we design and run two user surveys: with experts in the domain of OSS supply chain security, and with software developers, which are heavy consumers of OSS.

### A. Systematic Literature Review

The SLR accomplishes two goals. First, through exploring the state-of-the-art of OSS supply chain security, we identify and specify the abovementioned research questions. Second, it supports identifying and collecting relevant attack vectors and suitable safeguards. The SLR itself follows a three step methodoloy comprising *planning, conducting*, and *reporting* [23] [24] depicted in Figure 2 and described hereafter.

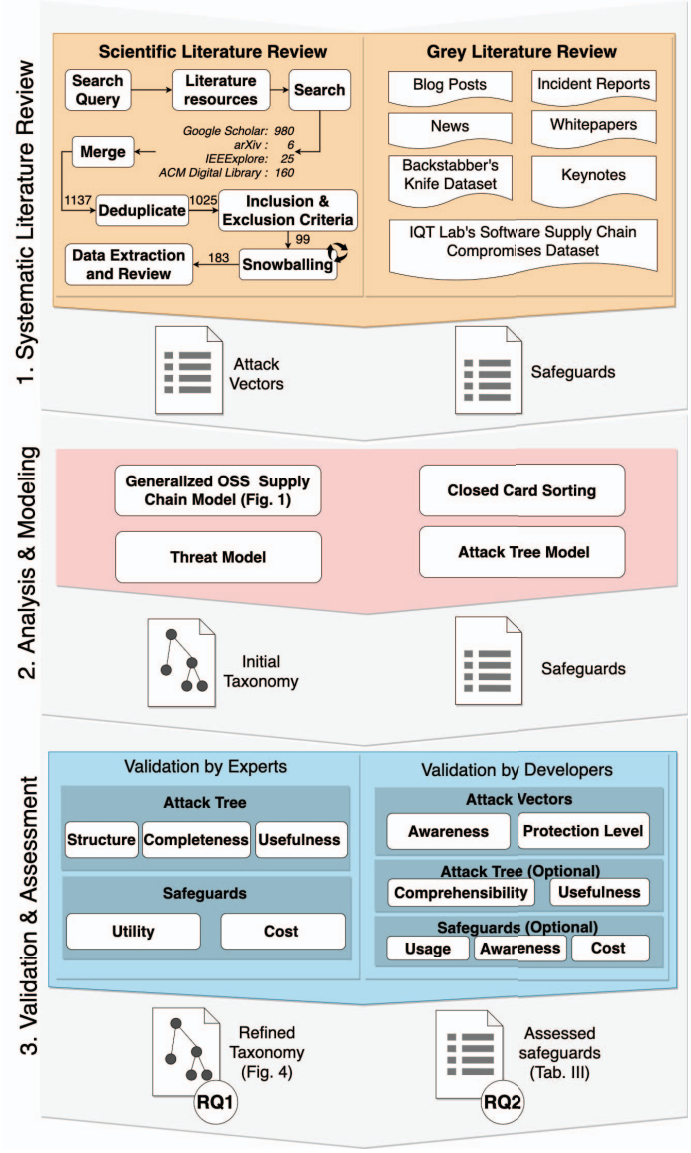


Fig. 2: Our methodology comprises a literature review, the modeling of taxonomy & safeguards, and the validation.

*Search Strategy:* This step defines the search terms, the query used on the identified resources, and the inclusion criteria. For our purpose, we used the following query to search for the terms anywhere in the documents:

```
("open source" OR "open-source" OR "OSS" OR "free"
OR "free/libre" OR "FLOSS) AND "software" AND
("supply chain" OR "supply-chain") AND ("security"
OR "insecurity" OR "attack" OR "threat"
OR "vulnerability")
```

The four digital libraries used to collect the primary studies are[2]: Google Scholar (980 results), arXiv (6), IEEExplore (25) and ACM Digital Library (160). After removing duplicates from the total of 1171 search results, 1025 papers remained.

We only included peer-reviewed articles in journals and conferences, technical reports, and Ph.D./Master theses written in English and published before March 2022. Also, we only included studies related to security aspects, threats and malware in the areas of OSS development, VCS, build systems and package repositories, as well as malware detection and software supply chain security. The application of those inclusion criteria reduced the 1025 results obtained in the previous phase to 99 papers. Discarded documents concern security aspects in physical or hardware supply chains, or general discussions about emerging technologies (of which OSS security is an example).

We then applied the *snowballing technique* on all the remaining works to find resources missed during the initial search, thereby applying the same inclusion criteria. This resulted in the addition of another 84 new studies.

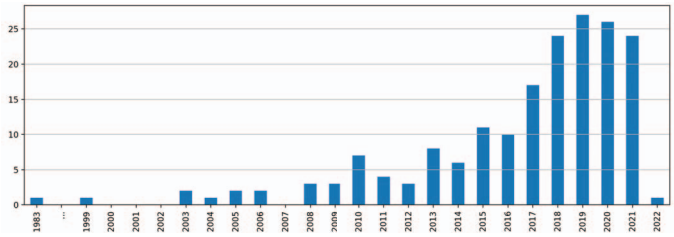


Fig. 3: No. selected scientific articles per year of publication.

*Data Extraction:* The selection process resulted in a total of 183 scientific works, mostly from the last few years (cf. Figure 3), which were carefully reviewed to extract information about common threats, attack vectors, and related safeguards. The complete list of the selected works is accessible online [3].

### B. Grey Literature

In addition to scientific literature, especially to cover as many real-world attacks and vulnerabilities as possible, we looked at grey literature like blog posts, whitepapers, or incident reports. To this end, we periodically reviewed several news aggregators and blogs (cf. Appendix B). Also, we used the same search query as in Section III-A for searching on Google. All results were filtered using the selection criteria from Section III-A, and the *snowballing technique* was applied to further extend the set of sources.

### C. Analysis and Modeling of the Attack Scenario

We perform the analysis of the OSS supply chain depicted in Figure 1 to classify the identified attack vectors during the SLR. Then we model such attacks using the semantic of attack trees. The goal of these two steps is to answer to **RQ1.2**, i.e., propose a taxonomy of OSS supply chain attacks.

The analysis of the attack scenario in the context of OSS development started from the identification of the stakeholders (i.e., *actors*), *systems*, as well as their relationship (i.e., *channels*). We have described such elements in Section II and depicted in Figure 1. This analysis was useful to identify potential categories to structure the identified attack vectors.

During the modeling phase, we adopted an *attack-centric* methodology whose purpose is to characterize the hostility of

[2]Their URLs can be found in Appendix B

[3]https://doi.org/10.5281/zenodo.6395965

the environment and the attack complexity for exploiting a system vulnerability [25]. In particular, we performed closed card-sorting in the form of a tree-test intending to build a taxonomy of OSS supply chain attacks as an attack tree. Closed card-sorting is an information architecture technique taken from User eXperience (UX) design, in which the participants are asked to structure a given set of information [26]. A tree-test is a particular case of a card-sorting problem, where the information is structured in a tree.

For the attack tree modeling, we used as a starting point the attack tree proposed by Ohm et al. [19], whose root node is *Injection of Malicious Code (into dependency tree)*. Thanks to a rigorous structure, deeper refinement and the SLR, we identified many additional attack vectors (107 instead of 19).

The main criteria to structure the attack tree were: degree of interference with existing ecosystems (1st-level nodes), stages of the software supply chain (i.e., source, build, distribute), and the system and stakeholders involved in each stage.

The initial naming and arrangement have been changed to reflect the expert feedback described in Section IV-B. The refined version of our initial attack tree is depicted in Figure 4.

### D. Identification and Classification of Safeguards

To identify general safeguards, also in this case we reviewed the scientific and grey literature described in Section III-B. Then, each safeguard is classified according to control type, stakeholder involvement, and mitigated attack vector(s).

**Control type** classification follows the well-known distinction of directive, preventive, detective, corrective, and recovery controls [27]. However, since our focus is on *how* malicious code – no matter its actual intent – can be injected into open-source and corresponding safeguards explains why recovery controls were out of scope.

**Stakeholder involvement** reflects which role(s), maintainers, system administrators or consumers, can or must become active to effectively implement a given control.

Finally, each safeguard has been assigned to those node(s) of the attack tree that it mitigates (partially or fully). To reflect the broader or narrower scope, they were assigned to the tree node with the least possible depth.

### E. Survey Methodology

We conducted two online surveys targeting two different audiences. First, we addressed experts in the domain of software supply chain security to validate the proposed taxonomy of attack vectors (**RQ1**) and to collect feedback regarding the utility and costs of safeguards (**RQ2.2**). Second, we addressed developers to rate their use of attack vectors and perceived protection level. Optionally, they could additionally assess the taxonomy and the use and awareness of safeguards from the perspective of open-source consumers (**RQ2.3**).

*Questionnaire Design and Development:* We conducted a cross-sectional survey [28] consisting of the following four parts.

*Demographics:* This part collects background information about survey participants, especially their skillset to check whether our objectives to address security experts and developers are met, but also programming languages used, or whether they actively participate in OSS projects. The results are discussed in Section VII.

*Taxonomy:* In the expert survey, this part was meant to validate and assess the proposed taxonomy. Before displaying our proposed taxonomy in its entirety, we used tree-testing [29] to capture how easily users find tree nodes. This helped validate the nodes' parent-child relationships. Afterwards, participants were asked to explore an interactive visualization of the complete taxonomy, and then to rate its structure, node names, coverage, and its usefulness (to support different use-cases) on a Likert scale from 1 (low) to 5 (high).

In the developer survey, this part started with a presentation of the taxonomy's first-level nodes, including attack vector names and descriptions. Participants were asked whether they are aware of such attacks and whether they – or their organization – use any mitigating safeguards. Optionally, participants could continue this part to explore the taxonomy and rate its comprehensibility and usefulness.

*Safeguards:* In the expert survey, the participants assessed the utility and costs of the selected safeguards. To this end, they were grouped by and presented according to the stakeholder roles involved in their implementation.

This entire part was optional in the developer survey. When opting-in, respondents only rated safeguards relevant according to their role in open-source projects (if any). When shown, survey participants provided feedback whether they use a given safeguard and its perceived costs (Likert scale).

*Pilot Survey and Pretest:* Interviews with two experts in user research and UX provided us feedback on the suitability and understandability of the survey. Their main suggestions were to shorten texts and improve content presentation, esp. of the tree-testing content. After implementing their feedback, we performed a pretest of the expert survey with 37 researchers from academia (i.e., Ph.D. students, researchers, and professors), and the developer survey with 14 master students. The feedback received from this pretest suggested further shortening texts, improving some questions, and adjusting the appearance of buttons.

*Sampling:* For the selection of participants in both questionnaires, we adopted the snowball sampling [30] technique. It consisted of inviting an initial group of participants, which were asked to further share the invitation in their appropriate network of knowledge. Due to this sampling technique, it is not possible to compute the response rate.

The initial list of domain experts was composed of authors of works analyzed during the SLR, as well as experts from industry and academia from our network. We included experts who performed relevant works in the context of OSS supply chain security (e.g, scientific publications, initiatives/projects of software foundations or industry). Similarly, the initial list of software developers has been created starting from our network of knowledge of practicioners.

The channels used to reach the participants have been emails, Linkedin, and direct recruitment during presentations.

The expert survey campaign began on 22 July 2021, the one for developers on 19 October 2021. Both questionnaires were closed for analysis on 24 November 2021, and reached a total of 17 and 134 respondents respectively.

*Survey Procedure and Data Protection:* Rather than using existing survey tools or services, we developed a custom solution. SurveyJS[4] was used to design the survey structure and content, which we exported as JSON file. This file was hosted using GitHub Pages, together with SurveyJS' runtime library and other resources. Participant answers were sent to a custom Google AppScript, which stored them in a Google spreadsheet. Answers were sent after each survey page and grouped using a random number generated in the beginning.

Applying the principle of data minimization, we did not collect IP addresses, names or other Personally Identifiable Information (PII). We also did not have access to 3rd party server logs. Moreover, the decoupling of survey frontend and backend made that the first 3rd party service provider only knows survey structure and content, while the second only sees (encoded) answers without understanding their semantics.

## IV. Attack Taxonomy and its Validation

This section presents the taxonomy built from 107 unique attack vectors collected through the review of scientific and grey literature. Following, it summarizes the results of its validation by domain experts, and the responses of software developers regarding problem awareness, understandability, and usefulness of our taxonomy.

### A. Taxonomy of Attacks on OSS Supply Chains

The attackers' high-level goal is to conduct a supply chain attack by injecting malicious code[5] into an OSS project such that it is downloaded by downstream consumers, and executed upon installation or at runtime. They can target any kind of project (e.g., libraries or word processors), direct or indirect downstream consumers, as many as possible, or very specific ones. The latter is possible by conditioning the execution of malicious code, e.g., on the lifecycle phase (install, test, etc.), application state, operating system, or properties of the downstream component it has been integrated into [19].

The entire taxonomy unfolding below this high-level goal is depicted in Figure 4 and summarized hereafter, whereby the 1st-level child nodes of the tree reflect different degrees of interference with existing packages.

**Develop and Advertise Distinct Malicious Package from Scratch** covers the creation of a new OSS project, with the intention to use it for spreading malicious code from the beginning or at a later point in time. Besides creating the project, the attacker is required to advertise the project to attract victims. Real-world examples affect PyPI, npm, Docker Hub or NuGet [19], [59]–[65].

**Create Name Confusion with Legitimate Package** covers attacks that consist of creating project or artifact names that resemble legitimate ones, suggest trustworthy authors, or play with common naming patterns. Once a suitable name is found, the malicious artifact is deployed, e.g., in a source or package repository, in the hope of being consumed by downstream users. As the deployment does not interfere with the resources of the project that inspired the name (e.g., legitimate code repository, maintainer accounts) the attack is relatively cheap.

Child nodes of this attack vector relate to sub-techniques applying different modifications to the legitimate project name: *Combosquatting* [74] adds pre or post-fixes, e.g., to indicate project maturity (`dev` or `rc`) or platform compatibility (`i386`). *Altering Word Order* [74] re-arranges the word order (`test-vision-client` vs. `client-vision-test`). *Manipulating Word Separators* [74] alters or adds word separators like hyphens (`setup-tools` vs. `setuptools`). *Typosquatting* [5], [15], [19], [71], [74], [75] exploits typographical errors (`dajngo` vs. `django`). *Built-In Package* [74] replicates well-known names from other contexts, e.g., built-in packages or modules of a programming language (`subprocess` for Python). *Brandjacking* [164] creates the impression a package comes from a trustworthy author (`twilio-npm`). *Similarity Attack* [165] creates a misleading name in a way different from the previous categories (`request` vs. `requests`).

**Subvert Legitimate Package** covers all attacks aiming to corrupt an existing, legitimate project, which requires compromising one or more of its numerous resources depicted in Figure 1. As a result, this subtree is much larger compared to the previous ones, esp. because subtrees related to user and system compromises occur multiple times in the different supply chain stages. The remainder of this section is dedicated to sub-techniques of this first-level node.

*Inject into Sources of Legitimate Package:* It relates to the injection of malicious code into a project's codebase. For the attacker, this has the advantage to affect all downstream users, no matter whether they consume sources or pre-built binary artifacts (as part of the codebase, the malicious code will be included during project builds and binary artifact distribution).

This vector has several sub-techniques. Taking the role of contributors, attackers can use *hypocrite merge requests* to turn immature vulnerabilities into exploitable ones [11], or exploit IDE rendering weaknesses to hide malicious code, e.g., through the use of Unicode homoglyphs and control characters [10], or the hiding and suppression of code differences [166]. To *contribute as maintainer* requires to obtain the privileges necessary for altering the legitimate project's codebase, which can be achieved in different ways. Using Social Engineering (SE) techniques on legitimate project maintainers [167], [168], by *taking over legitimate accounts* (e.g., reusing compromised credentials [169]), or by *compromising the maintainer system* (e.g., exploiting vulnerabilities [113]). The latter can also be achieved through a malicious (OSS) component, e.g., IDE plugin, which is reflected through a recursive reference to the root node.

The legitimate project's codebase can also be altered by *tampering with its VCS*, thus, bypassing a project's established contribution workflows. For instance, by compromising system user accounts [115], [116], or by exploiting configuration/-

[4]https://surveyjs.io/

[5]This does not only cover the addition of program code but malicious changes in general, e.g., the introduction of new malicious dependencies or the (re)introduction of vulnerabilities, e.g., the removal of authorization checks.

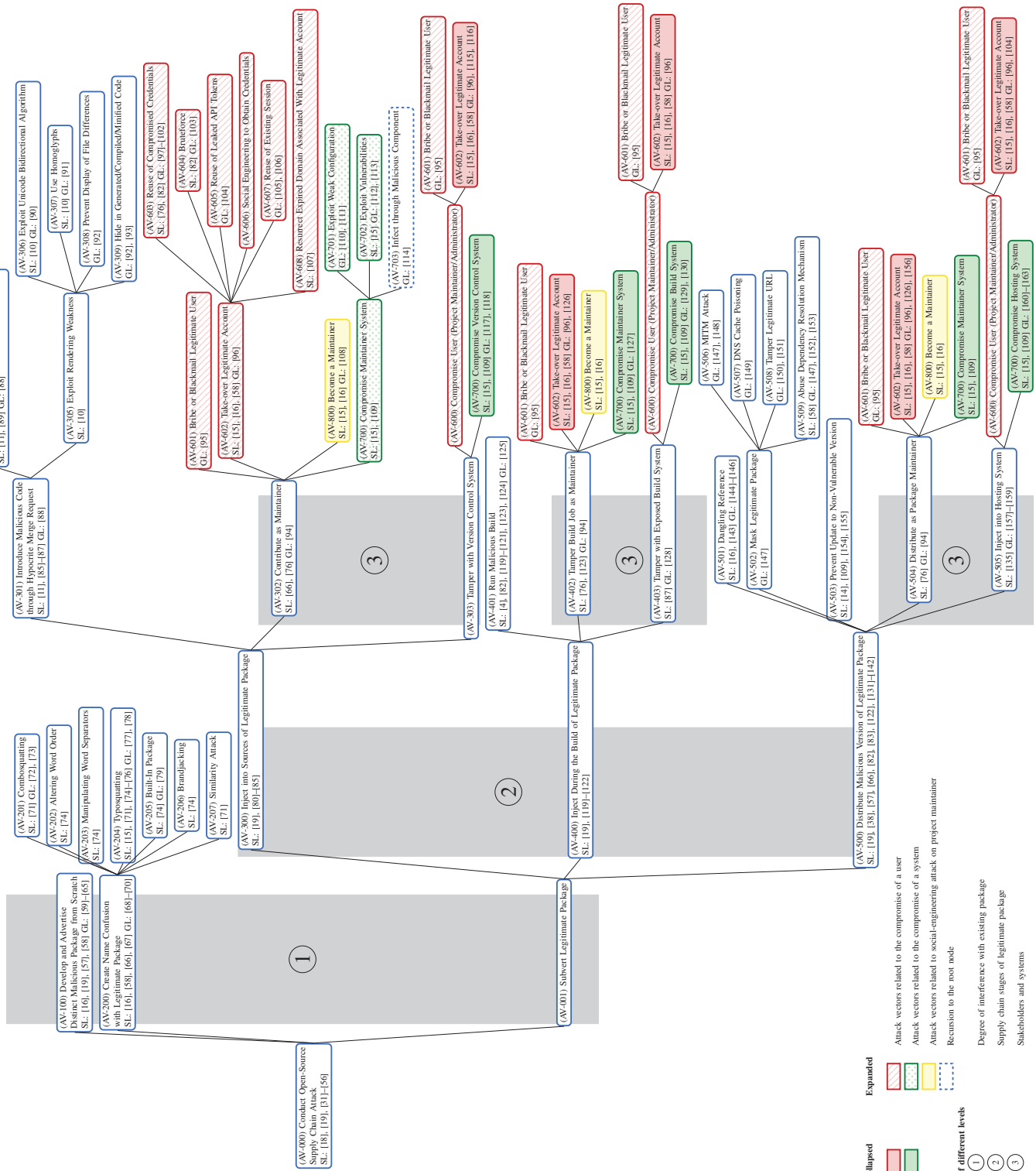


Fig. 4: Refined version of the taxonomy for OSS supply chain attacks. It takes the form of an attack tree with the attacker's top-level goal to inject malicious code into open-source project artifacts consumed and executed by downstream users. This version reflects the feedback of 17 domain experts on the initial version, collected through an online survey. Subtrees for user and system compromises exist multiple times, only their first occurrence is expanded. The grey, numbered rectangles illustrate the different criteria used for structuring the tree. Each node has references to both Scientific Literature (SL) and Gray Literature (GL).

software vulnerabilities [118], [170], [171], an attacker could access the codebase in insecure ways.

*Inject During the Build of Legitimate Package:* Greatly facilitated by language-specific package managers like Maven or Gradle for Java, it became common to download pre-built components from package repositories rather than OSS project's source code from its VCS. Therefore, the injection of malicious code can happen during the build of such components before their publication [6], [119], [120]. Though the spread is limited compared to injecting into sources, the advantage for the attacker is that the detection of malicious code inside pre-built packages is typically more difficult, especially for compiled programming languages. One sub-technique is *running a malicious build job* to tamper with system resources shared between build jobs of multiple projects [123] (e.g., the infection of Java archives in NetBeans projects [125]). An attacker can also *tamper the build job as maintainer*, e.g., by taking over legitimate maintainer accounts, becoming a maintainer, or compromising their systems (cf. XCodeGhost malware [127]). Similarly, the attacker could comprise build systems, esp. online accessible ones, e.g., by compromising administrator accounts [126] or exploiting vulnerabilities [129], [172].

*Distribute Malicious Version of Legitimate Package:* Pre-built components are often hosted on well-known package repositories like PyPI or npm, but also on less popular repositories with a narrower scope. In addition, the components can be mirrored remotely or locally, made available through CDNs (e.g., in the case of JavaScript libraries), or cached in proxies. This attack vector and its sub-techniques cover all cases where attackers tamper with mechanisms and systems involved in the hosting, distribution, and download of pre-built packages.

*Dangling references* (re)uses resource identifiers of orphaned projects [144]–[146], e.g., names or URLs. *Mask legitimate package* [147] targets package name or URL resolution mechanisms and download connections. Their goal is the download of malicious packages by compromising resources external to the legitimate project. This includes Man-In-The-Middle (MITM) attacks, DNS cache poisoning, or tampering with legitimate URLs directly at the client [173]. Particularly, package managers follow a (configurable) resolution strategy to decide which package version to download, from where, and the order of precedence when contacting multiple repositories. Attackers can *abuse such resolution mechanisms* and their configurations [152], [174]. Attackers can also *prevent updates to non-vulnerable versions* by manipulating package metadata [154], e.g., by indicating an unsatisfiable dependency for newer versions of a legitimate package. Finally, the involvement of systems and users in package distribution results in attack vectors similar to previous ones. Attackers can take the role of legitimate maintainers, thus, *distribute as maintainer*, e.g., by taking over package maintainer accounts (e.g., `eslint` [156]), the second most common attack vector after typosquatting [19]. They can also compromise maintainer systems, or directly *inject into the hosting system*, e.g., by compromising administrator accounts [175] or exploiting vulnerabilities [160]–[162], [176].

**Response to RQ1.1**: Through the review of 183 scientific papers as well as grey literature, we identified and generalized 107 unique attack vectors on OSS supply chains, supported by 94 real-world attacks or vulnerabilities.

### B. Validation and Assessment by Domain Experts

The initial version of the taxonomy was validated and assessed by **17 domain experts**. Their feedback has been retrofitted resulting in the taxonomy depicted in Figure 4.

*Validation:* This section reports expert feedback on the comprehensiveness of attack vectors, and the correctness, comprehensibility, and usefulness of the taxonomy.

Before having seen the taxonomy in its entirety, the tree-testing required experts to assign attack vectors to the first level nodes of the initial taxonomy. Over a total of 311 assignments by all experts, 234 (75%) matched the structure of the initial taxonomy, while 77 (25%) did not, which shows an overall agreement on the structure.

Following, the experts were presented with the initial version of the entire taxonomy, and asked to assess different qualities using a Likert scale ranging from 1 (low) to 5 (high).

14 (82%) experts agreed to the overall structure with a rating of 4 or 5, slightly higher compared to the results of the tree-test. This could be due to some node names not being self-explanatory enough when shown with too little context.

Experts were further asked to rate the correctness of the taxonomy's 1st-level nodes in regards to naming, tree location, and sub-tree structure. All the first-level nodes received an overall good agreement with naming, categorization, and sub-tree structure, except *Develop and Advertise Distinct Malicious Package from Scratch*. The latter only received neutral feedback on its sub-tree, a light agreement with its categorization, and a clear disagreement with its initial naming.

12 (71%) of the experts agreed with the **completeness of the attack tree**.

*Usefulness and Use-Cases:* In this part of the questionnaire, experts rated the usefulness and possible use cases of the proposed taxonomy.

15 (88%) rated the usefulness of the taxonomy to *understand the attack surface of the OSS supply chain* with a 4 or 5. Fewer experts considered it being useful to understand *attacker tactics and techniques* (12 (71%)) or *attackers' cost/benefits considerations* (5 (29%)).

Regarding the expert options about possible uses of the proposed taxonomy, the Top-3 use-cases are *threat modeling*, *awareness and training* and *risk assessment*. Another possible use-case, though not included in the survey, is to scope penetration tests.

### C. Validation and Assessment by Developers

The initial version of the taxonomy has also been validated and assessed by **134 software developers** in regards to the awareness of main attack vectors (1st-level taxonomy nodes), whether those are mitigated (by themselves or their organization), and – optionally – the understandability and utility of the taxonomy.

*Awareness about Attack Vectors:* The awareness of main attack vectors ranged from 120 (90%) for *Develop and Advertise Distinct Malicious Package from Scratch* to 86 (64%) for *Inject During the Build of Legitimate Package*.

For all but one vector, the majority of respondents answered not to know whether they are protected. Only for *Develop and Advertise Distinct Malicious Package from Scratch*, the majority believes in being protected (52%). For both vectors *Inject During the Build* and *Distribute Malicious Version*, 19 (14%) respondents were sure that no protection exists.

*Taxonomy Understandability and Utility Assessment:* Among the 134 participants, 53 (40%) decided to perform the optional assessment of the taxonomy's understandability and utility to understand the supply chain's attack surface. Considering a rating of 4 or 5, 41 (77%) found the taxonomy understandable and 46 (87%) recognized it as a useful means to create awareness.

> **Response to RQ1.2**: The proposed taxonomy of attacks on OSS supply chains takes the form of an attack tree covering all 107 vectors identified beforehand. Its validation by 17 domain experts and 134 software developers showed overall agreement with structure and naming, comprehensiveness, comprehensibility, and suitability for use-cases like threat modeling, awareness, training, or risk assessment.

## V. Safeguards and their Assessments

Subsection V-A starts with a short overview about safeguards against OSS supply chain attacks, which were identified through literature review and generalized to become agnostic of specific prg. languages or ecosystems. Subsections V-B and V-C report the results of the two surveys conducted with domain experts and software developers to validate and assess the safeguards regarding different qualities, e.g., utility or cost.

### A. List of Safeguards

In total, we identified 33 safeguards that partially or completely mitigate the before-mentioned attack vectors. Both implementation and use of those safeguards can incur non-negligible costs, also depending on the specifics of prg. languages and ecosystems at hand. Thus, the selection, combination and implementation of safeguards require careful planning and design, to balance required security levels and costs.

The complete list of safeguards can be found in Table II of Appendix A, including a classification after control type. All safeguards are mapped to the vector(s) they mitigate, some to the top-level goal due to (partially) addressing all vectors (e.g., establishing a vetting process), others to more specific subgoals. Some safeguards can be implemented by one or more stakeholders, while others require the involvement of multiple ones to be effective (e.g., signature creation and verification).

**Common Safeguards** comprises 4 countermeasures that require all stakeholders to become active, i.e., project maintainers, open-source consumers, and administrators (service providers). For example, a detailed Software Bill of Materials (SBOM) has to be produced and maintained by the project maintainer [8], ideally using automated Software Composition Analysis (SCA) tools. Following, the SBOM must be securely hosted and distributed by package repositories, and carefully checked by downstream users in regards to their security, quality, and license requirements.

**Safeguards for Project Maintainers and Administrators** comprises eight safeguards. *Secure authentication*, for instance, suggests service providers to offer Multi-Factor Authentication (MFA) or enforce strong password policies, while project maintainers should follow authentication best-practices, e.g., use MFA where available, avoid password reuse, or protect sensitive tokens.

**Safeguards for Project Maintainers** includes seven countermeasures. Generally, OSS projects use hosted, publicly accessible VCSs. Maintainers should then, e.g., conduct careful *merge request reviews* or enable *branch protection rules* for sensitive project branches to avoid malicious code contributions. As project builds may still happen on maintainers' workstations, they are advised to use *dedicated build services*, esp. *ephemeral environments* [9]. Additionally, they may *isolate build steps* [123] such that they cannot tamper with the output of other build steps.

**Safeguards for Administrators and Consumers** comprises five countermeasures. For example, both package repository administrators and consumers can opt for *building packages directly from source code* [177], rather than accepting pre-built artifacts. If implemented by package repositories, this would reduce the risk of subverted project builds. If implemented by consumers, this would eliminate all risks related to the compromise of 3rd-party build services and package repositories, as they are taken out of the picture.

**Safeguards for Consumers** includes nine countermeasures that may be employed by the downstream users. The consumers of OSS packages may reduce the impact of malicious code execution when consuming by *isolating the code and/or sandboxing* it. Another example is the *establishment of internal repository mirrors* [178] of vetted components.

> **Response to RQ2.1**: We identified 33 general safeguards to be used by the different stakeholders, mostly detective or preventive ones, and mapped them to the node(s) of the attack tree they mitigate partially or fully.

### B. Experts Validation and Assessment

This section presents the feedback of 17 experts regarding the safeguards' utility to mitigate risks, and their associated costs for implementation and continued use.

In summary, almost all the safeguards received medium to high utility ratings, while the cost ratings range from low (i.e., minimum mean value of 2.0) to very high (i.e., maximum mean value of 4.8).

Table I provides all feedback collected for the 33 safeguards, following a discussion of safeguards with the highest, re-

spectively lowest Utility-to-Cost (U/C) ratios, and some other interesting cases.

**High U/C ratio.** Both *Protect production branch*, *Remove un-used dependencies* and *Version pinning* show the highest U/C ratio, thus, are considered to be useful and cheap controls. The use of *Resolution Rules* also shows a good U/C ratio, even though one expert highlighted that "very few projects" use them, and that the implementation would require the modification of all package managers. On average, *Preventive Squatting* only received neutral ratings (3.1 for utility and 2.9 for cost) and also raised some concerns: two of the experts highlighted that it could be good to "try to prevent name squatting, but hard to fully enforce" also due to legitimate reasons for similar names (e.g., to help consumers identify package relationships).

**Low U/C ratio.** *Build Dependencies from Sources*, reportedly used by Google [177], received a very low utility rating (mean and median of 3.0) and overall the lowest U/C ratio. Considering that its use would prune the subtrees of both vectors *Inject During the Build* and *Distribute Malicious Version*, we expected a higher utility rating. One expert claimed that "building from source only helps if someone scans and reviews the code". Possibly referring to flaky builds [179], another expert highlighted that "rebuilding software from source can sometimes introduce problems".

*Merge Request Reviews* received the highest average utility rating (4.6), which could be because if malicious code is injected into the sources, it is guaranteed to arrive at consumers, no matter how they consume it.

*Reproducible Builds* received a very high utility rating (5) from 10 participants (58.8%), but also a high-cost rating (4 or 5) from 12 (70.6%). One expert commented that a "reproducible build like used by Solarwinds now, is a good measure against tampering with a single build system" and another claimed this "is going to be the single, biggest barrier".

*Scoped Packages*, proposed as an effective safeguard against *Abuse of Dependency Resolution mechanisms* [178], [180], mostly received neutral ratings (3) for both utility and cost.

**Response to RQ2.2**: We have qualitatively assessed the utility and costs of the 33 safeguards by surveying 17 experts. The three safeguards *Protect production branch*, *Remove un-used dependencies* and *Version pinning* showed the best U/C ratio while *Build dependencies from sources* showed the worst.

### C. Developers Validation and Assessment

In this optional part of the survey, developers were asked to assess the usage and costs of a subset of safeguards that were selected according to the stakeholders' roles exercised in their daily work (collected in the demographic part). Among the total of 134 respondents, 30 assessed the *Common Safeguards*, 5 the *Safeguards for Project Maintainers*, 4 the *Safeguards for Maintainers and Administrators*, 24 the *Safeguards for Administrators and Consumers*, and 22 the *Safeguards for Consumers*. Complete results are shown in Table I

*Remove un-used dependencies* is frequently used by developers, which contrasts with the observations of Soto-Valerio et al. [181], who found that many Java projects had bloated (unused) dependencies. Other countermeasures that appear to be widely used among the respondents are *Version pinning* and *Open-source vulnerability scanners*, the latter of which does not only address attacks, but also the use of dependencies with known vulnerabilities.

Concerning the attack vector *Create Name Confusion*, where 70% of the developers claimed to be aware of the problem, we can observe that corresponding safeguards *Typo guard/Typo detection* and *Preventive squatting the released package* are only used by a minority of respondents.

It is also noteworthy to mention that developers' cost ratings generally coincide with those of the domain expert. Surprising exceptions are *Application Security Testing* and *Enstablish vetting process for Open-Source components hosted in internal/public repositories*, both having a median of 3 from developers, compared to a median of 5 from experts.

**Response to RQ2.3**: 134 software developers provided feedback on the use of safeguards. The three most-used ones are *Remove un-used dependencies*, *Version pinning* and *Integrate Open-Source vulnerability scanner into CI/CD pipeline*.

## VI. Discussion

While the taxonomy presented in Section IV is largely agnostic to ecosystems, this section discusses differences between ecosystems and highlights possible future research on the basis of our work.

*1) Differences between Ecosystems:* As mentioned in Section II-E, the attacker's high-level goal is to **inject malicious code** into open-source artifacts such that it is executed downstream. Several techniques to this end are indeed independent of specific ecosystems/languages, e.g., *Take-over Legitimate Account* or *Become Maintainer*.

Other attack vectors, however, are specific: *Abuse Dependency Resolution Mechanism* attacks depend on the approach and strategy used by the respective package manager to resolve and download declared dependencies from internal and external repositories. For instance, Maven, npm, pip, NuGet or Composer were affected by the dependency confusion attack, while Go and Cargo were not [180]. Several attacks below *Exploit Rendering Weakness* depend on the interpretation and visualization of (Unicode) characters by user interfaces and compiler/interpreters [10]. Also name confusion attacks need to consider ecosystem specificities, esp. *Built-In Packages*.

More differences exist when it comes to the **execution or trigger of malicious code**, which is beyond the taxonomy's primary focus on code injection. For Python and Node.js, this is commonly achieved through installation hooks, which trigger the execution of code provided in the downloaded package (e.g., in `setup.py` for Python or `package.json` for JavaScript). A comparable feature is not present in most

| Safeguard | Experts | | | | | | | Developers | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Utility | | | Cost | | | | | Cost | |
| | Mean | Median | | Mean | Median | | Mean U/C | Usage | Mean | Median |
| Protect production branch [85], [86] | 4.2 | 4.0 | | 2.0 | 2.0 | | **2.10** | Y N | 1.8 | 2.0 |
| Remove un-used dependencies [181] | 4.3 | 5.0 | | 2.1 | 2.0 | | 2.05 | Y N | 2.0 | 2.0 |
| Version pinning [15], [178], [180] | 3.7 | 3.0 | | 2.2 | 2.0 | | 1.68 | Y N | 2.1 | 2.0 |
| Dependency resolution rules | 4.1 | 4.0 | | 2.6 | 3.0 | | 1.58 | Y N | 2.7 | 3.0 |
| User account management [135] | 3.9 | 4.0 | | 2.6 | 3.0 | | 1.50 | Y N | 2.3 | 2.5 |
| Secure authentication (e.g., MFA, password recycle, session timeout, token protection) [15], [66] | 4.3 | 5.0 | | 2.9 | 3.0 | | 1.48 | Y N | 2.5 | 3.0 |
| Use of security, quality and health metrics [40] | 3.5 | 4.0 | | 2.6 | 3.0 | | 1.35 | Y N | 2.7 | 3.0 |
| Typo guard/Typo detection [15], [182] | 3.9 | 4.0 | | 2.9 | 4.0 | | 1.34 | Y N | 3.1 | 3.0 |
| Use minimal set of trusted build dependencies in the release job [123] | 4.1 | 4.0 | | 3.1 | 3.0 | | 1.32 | Y N | 3.8 | 4.0 |
| Integrity check of dependencies through cryptographic hashes [9], [36], [83], [109], [131], [135], [138] | 3.3 | 3.0 | | 2.5 | 2.0 | | 1.32 | Y N | 2.3 | 2.0 |
| Maintain detailed SBOM [5], [8], [53], [183], [184] and perform SCA [8], [31], [43], [48], [51], [53], [55], [56] | 4.2 | 5.0 | | 3.4 | 4.0 | | 1.24 | Y N | 2.9 | 3.0 |
| Ephemeral build environment [9], [123] | 3.6 | 3.0 | | 2.9 | 3.0 | | 1.24 | Y N | 2.8 | 2.5 |
| Prevent script execution | 3.7 | 3.0 | | 3.0 | 3.0 | | 1.23 | Y N | 2.4 | 2.0 |
| Pull/Merge request review [86] | 4.6 | 5.0 | | 3.8 | 4.0 | | 1.21 | Y N | 3.6 | 4.0 |
| Restrict access to system resources of code executed during each build steps [42], [123], [185] | 4.0 | 4.0 | | 3.3 | 3.0 | | 1.21 | Y N | 3.8 | 3.5 |
| Code signing [47], [83], [109], [135], [138], [141], [155] | 3.7 | 4.0 | | 3.1 | 3.0 | | 1.19 | Y N | 3.1 | 3.0 |
| Integrate Open-Source vulnerability scanner into CI/CD pipeline | 3.8 | 4.0 | | 3.3 | 3.0 | | 1.15 | Y N | 3.1 | 3.0 |
| Use of dedicated build service [9] | 3.6 | 4.0 | | 3.3 | 3.0 | | 1.09 | Y N | 3.0 | 3.0 |
| Preventive squatting the released packages | 3.1 | 3.0 | | 2.9 | 3.0 | | 1.07 | Y N | 3.8 | 3.5 |
| Audit, security assessment, vulnerability assessment, penetration testing | 4.3 | 4.0 | | 4.1 | 4.0 | | 1.05 | Y N | 3.8 | 3.5 |
| Reproducible builds [121], [136], [186] | 4.2 | 5.0 | | 4.1 | 4.0 | | 1.02 | Y N | 3.5 | 4.0 |
| Isolation of build steps [123] | 3.1 | 3.0 | | 3.1 | 3.0 | | 1.00 | Y N | 3.2 | 3.0 |
| Scoped packages [178], [180] | 2.9 | 3.0 | | 2.9 | 3.0 | | 1.00 | Y N | 2.8 | 2.0 |
| Establish internal repository mirrors and reference one private feed, not multiple [178] | 3.6 | 3.0 | | 3.7 | 4.0 | | 0.97 | Y N | 2.7 | 3.0 |
| Application Security Testing [34], [39], [41], [46], [55], [56], [58], [66], [80], [122], [134], [187] | 4.1 | 4.0 | | 4.3 | 5.0 | | 0.95 | Y N | 3.7 | 3.0 |
| Establish vetting process for Open-Source components hosted in internal/public repositories [15], [16], [32], [134], [188] | 4.1 | 4.0 | | 4.3 | 5.0 | | 0.95 | Y N | 3.8 | 3.5 |
| Code isolation and sandboxing [42], [57], [185] | 3.9 | 4.0 | | 4.2 | 4.0 | | 0.93 | Y N | 3.2 | 3.0 |
| Runtime Application Self-Protection | 3.7 | 4.0 | | 4.2 | 4.0 | | 0.88 | Y N | 3.8 | 4.0 |
| Manual source code review [66] | 4.1 | 4.0 | | 4.8 | 5.0 | | 0.85 | Y N | 4.4 | 5.0 |
| Build dependencies from sources | 3.0 | 3.0 | | 4.1 | 4.0 | | 0.73 | Y N | 3.8 | 4.0 |

TABLE I: Assessment of safeguards by 17 domain experts (left) and 134 developers (right). Utility and cost assessments were given on a Likert scale, the numbers are shown with bar plots, from 1 (low) to 5 (high). The background of mean and median values are determined by the intervals $[1, 2.5]$, $(2.5, 3.5]$ and $(3.5, 5.0]$. Safeguards are shown in the order of the mean of their **Utility-to-Cost Ratio (U/C)** (descending). Developer feedback on safeguard use was collected with yes/no questions, the number of respective answers are shown using a bar plot.

compiled languages, like Java or C/C++. In such cases, execution is achieved either at runtime, e.g., by embedding the payload in a specific function or initializer, or by poisoning test routines [19].

Differences also exist in regards to **code obfuscation and malware detection**. In case of interpreted languages, downloaded packages contain the malware's source code, which makes it more accessible to analysts compared to compiled languages. The presence of encoded or encrypted code in such packages proofed being a good indicator of compromise [58], as there are few legimitate use-cases for open-source packages. Minification is one of them, however, matters primarily for frontend JavaScript libraries. Indeed, many existing attacks did not employ obfuscation or encryption [19] techniques. Still, the quantity of open-source packages and versions makes manual inspection very difficult, even if source code is accessible.

When it comes to compiled code, well-known techniques like packing, dead-code insertion or subroutine reordering [189] make reverse engineering and analysis more complex. It is also noteworthy that ecosystems for interpreted languages ship compiled code. For instance, many Python libraries for ML/AI use-cases include and wrap platform-specific C/C++ binaries.

For what concerns **safeguards**, several of them are specific to selected package managers, namely *Scoped packages* (Node.js) and *Prevent script execution* (Python and Node.js). All others are relevant no matter the ecosystem, however, control implementations and technology choices differ, e.g., in case of *Application Security Testing*. Duan et al. [15] present a comparative framework for security features of package repositories (exemplified with PyPI, npm and RubyGems).

*2) Benefits of the Taxonomy for Future Research and Open Challenges:* Our work systematizes knowledge about OSS supply chain security by abstracting, contextualizing and classifying existing works. The proposed taxonomy can benefit future research by offering a central point of reference and a common terminology. The comprehensive list of attack vectors and safeguards can support assessing the security level of open-source projects, e.g., to conduct comparative empiric studies across projects and ecosystems and over time.

An open challenge in OSS supply chain attacks is the detection of malicious code. The availability of source code in ecosystems for interpreted languages suggests that malware analysis is more straight-forward. Still, recent publications focus on those ecosystems, esp. JavaScript and Python [15], [16], [57], [58], [74], partly due to their popularity, but also because existing malware analysis techniques cannot be easily applied. More subtle attacks, such as intentional insertion of vulnerabilities, complicate detection since they require analysis of the context of the change to distinguish it from an accidentally introduced vulnerability [80]. Additionally, code generation and the difficulty in identifying VCS commits that correspond to pre-built components, as highlighted in [132], [133], make malware analyis of source code difficult. The safeguard *Reproducible builds* [121], [136], [186] addresses this problem, however, it is not commonly applied, considered costly (cf. Table I) and more complex projects require significant implementation efforts.

## VII. User Survey Demographics

This section provides demographic information about the respondents of the two online surveys. In summary, the respondents to the expert survey meet the requirement of being experts in the domain and participate actively in OSS projects. The respondents to the developer survey regularly consume OSS and have little knowledge of supply chain security.

*Domain Experts:* **17 respondents** participated in the online survey designed for experts in the domain of *software supply chain security*. According to the self-assessment of their skills, 12 respondents consider themselves **knowledgeable in the domain of supply chain security**, but also in software security (14) and development (12). Considering their acquaintance with 11 popular languages [190], the respondents cover 9 out of them, whereby Python, Java and JavaScript are covered best, while nobody had a background in .NET and Objective-C. 14 out of the 17 respondents are active participants in OSS projects, and were asked about their respective role (multiple choice): all 14 are contributors, 7 are project maintainers, and 3 exercise other roles. 9 experts work in the private sector, compared to 5 working in the public sector (e.g., government, academia) and 3 in the not-for-profit sector. They cover the industry sectors information industry (8), computer industry (2), telecommunications (2), entertainment industry (1), mass media (1), defense (1) and others (2).

*Developers:* **134 respondents** participated in the online survey designed for software developers, who were assumed to exercise the role of downstream consumers in OSS supply chains. This assumption was confirmed given that 121 (90%) responded to using open-source components in their daily job. Moreover, 37 (28%) actively participate in OSS projects: 31 as contributors and 22 as maintainers (multiple choice). 74 are also maintainers of code repositories, and 21 administer package repositories. The self-assessment of their skills shows that they are **knowledgeable in software development** (113), and less so in supply chain security (22) and software security (44). They cover all of the 11 programming languages (multiple choice), whereby Java, JavaScript and Python are the most popular ones. The majority of the respondents (120) work in the private sector. In terms of industry sectors, computer and information industry (55 and 54) outweigh other sectors.

## VIII. Related Works

In the following, we distinguish existing works related to specific aspects of OSS supply chains, e.g., technologies, systems, or stakeholder interactions, from more general ones covering the entire supply chain.

**Specific Works.** Giovanini et al. [167] leverage patterns in team dynamics to predict the susceptibility of OSS development teams to social engineering attacks. Gonzalez et al. [80] describe attacks aiming to inject malicious code in VCSs via commits. They propose a rule-based anomaly detector that uses commit logs and repository metadata to detect potentially malicious commits. In the same direction, Goyal et al. [191] analyze collaborative OSS development and highlight the problem of overwhelming information that potentially results in maintainers accepting malicious merge requests. Wheeler [192] describes the problem of code injection

into software by subverted compilers, and proposes Diverse Double-Compiling (DDC) to detect such attack. Within this context, Lamb et al. [186] propose an approach to determine the correspondence between binaries and the related source code through bit-for-bit checks of build processes, while Ly et al. [132] analyzed the discrepancies between Python code in a projects' VCS and its distributed artifacts. Gruhn et al. [123] analyze the security of CI systems, and identify web User Interface (UI)s and build processes as the main sources of malicious data. They propose a secure build server architecture, based on the isolation of build processes through virtualization. Multiple works address common threats to package managers of different ecosystems. Cappos et al. [14], [154] identify possible attack vectors related to a lack of proper signature management at the level of packages and their metadata, some of which we considered below *Distribute malicious version of legitimate package*. Zimmerman et al. [16] analyze security threats and associated risks in the npm ecosystem, and define several metrics describing the downstream reach of packages and maintainers, which allows identifying critical elements. Inversely, they also measure the number of implicitly trusted upstream packages and maintainers. Bagmar et al. [74] performed similar work for the PyPI ecosystem, and several of their vectors are subsumed below *Create name confusion with legit. package*. Duan et al. [15] propose a framework to qualitatively assess functional and security aspects of package managers for interpreted languages (i.e., Python, JavaScript, and Ruby). They provide an overview of stakeholders (and their relationships) in those package manager ecosystems, but do not specifically cover VCS and build systems. Also Kaplan et al. [66] present the state of the art of threats in package repositories and describe – also experimental – countermeasures from the scientific literature.

**General works.** Ohm et al. [19] manually inspect malicious npm, PyPi, and Ruby packages. They propose an attack tree – based on a graph of Pfretzschner et al. [46] – describing how to inject malicious code into dependency trees. The attack tree proposed in Section IV follows a more rigorous structure (degrees of interference with existing packages, supply chain stages, stakeholders and systems involved) and our SLR resulted in the addition of 89 attack vectors. Our results have been validated through two user surveys. Du et al. [184] describe a wide range of high-level software supply chain risks, both external (e.g., natural disaster, political factor) and internal ones (e.g., participants, software components). ENISA [193] proposes a taxonomy of supply chain attacks describing the techniques used by attackers and the targeted assets, both from the supplier and customer perspective. However, they only mention few high-level techniques. Torres-Arias et al. [83] propose in-toto, a framework based on the concepts of delegations and roles to cryptographically ensure the integrity of software supply chains through an end-to-end verification of each step and actors involved. Samuel et al. [155] propose The Update Framework (TUF) to overcome in-toto's main limitations regarding secure distribution, revocation and replacement of keys.

## IX. Threats to Validity

The taxonomy was modeled using the semantics of attack trees and several of its nodes reflect the characterizing stages of OSS supply chains, with code from project contributors and maintainers flowing to downstream consumers. Though its comprehensiveness, comprehensibility, and usefulness have been positively assessed by the survey participants, the taxonomy reflects the current state of the art. As the supply chain technologies evolve, it is expected that the proposed attack tree will evolve too.

We systematically reviewed the literature and continuously monitor aggregators of security news to create a comprehensive list of attack vectors, and collected feedback from domain experts to assess its completeness. Still, the complexity of OSS supply chains makes it very likely that new attack vectors and techniques will be discovered. The quality of the taxonomy will correspond to the degree of changes required to reflect such new attacks.

The feedback collected from survey participants could have been biased if we only considered experts that we directly know. Instead, thanks to the snowball sampling we have reached also people outside of our network. Considering authors of relevant scientific works, experts from academia and industry, all working in the specific area of software supply-chain security, allowed us to reach the intended audience (cf. Section VII): the 17 respondents of the expert survey were knowledgeable in supply chain security and actively participate in OSS projects, the 134 participants of the developer survey have knowledge in software development and use OSS regularly, and both groups cover a diverse range of prg. languages, incl. those subject to frequent attacks.

## X. Conclusion

As validated by domain experts, the proposed taxonomy of attacks on OSS supply chains is comprehensive, comprehensible, and serves different use-cases. It can benefit future research serving as a central reference point and setting a common terminology.

The listing of safeguards and their mapping to attack tree nodes helps to determine the exposure of given stakeholders to supply chain attacks. Their assessment in terms of utility and costs can serve to optimize the spending of limited security budgets. Future empiric studies may investigate the prevalence of the identified countermeasures, e.g. their use by given open-source projects. On our side, we aim at developing techniques for the detection of malicious code in compiled Java open-source components. Going forward, to raise awareness for threats to OSS supply chains, we will publish the interactive visualization of the taxonomy online. References to literature and real-world incidents will be kept up-to-date by using the open-source approach to help the taxonomy itself stay relevant. Finally, we would also like to put the taxonomy into practice for other use-cases, esp. risk assessment.

**Acknowledgements.** We thank all survey participants for their time and the constructive and insightful feedback. This work is partly funded by EU grants No. 830892 (SPARTA) and No. 952647 (AssureMOSS)

## REFERENCES


[1] S. Peisert, B. Schneier, H. Okhravi, F. Massacci, T. Benzel, C. Landwehr, M. Mannan, J. Mirkovic, A. Prakash, and J. B. Michael, "Perspectives on the solarwinds incident," *IEEE Security Privacy*, vol. 19, no. 2, pp. 7–13, 2021.

[2] European Network and Information Security Agency, "Enisa threat landscape 2021," 2021. [Online; accessed 20-October-2021].

[3] Joseph R. Biden JR., "Executive order on improving the nation's cybersecurity," 2021. [Online; accessed 20-October-2021].

[4] K. Thompson, "Reflections on trusting trust," *Commun. ACM*, vol. 27, p. 761–763, Aug. 1984.

[5] T. Herr, "Breaking trust–shades of crisis across an insecure software supply chain," 2021.

[6] B. Chess, F. D. Lee, and J. West, "Attacking the build through cross-build injection: how your build process can open the gates to a trojan horse," 2007.

[7] Sonatype, "Q3 2021 state of the software supply chain report," *URL: www. sonatype. com/resources/state-of-the-software-supply-chain-2021*, 2018.

[8] C. Clancy, J. Ferraro, R. Martin, A. Pennington, C. Sledjeski, and C. Wiener, "Deliver uncompromised: Securing critical software supply chains," *MITRE Technical Papers*, vol. 24, 01 2021.

[9] K. Lewandowski and M. Lodato, "Introducing slsa, an end-to-end framework for supply chain integrity," *URL: slsa. dev*, 2021.

[10] N. Boucher and R. Anderson, "Trojan source: Invisible vulnerabilities," 2021.

[11] Q. Wu and K. Lu, "On the feasibility of stealthily introducing vulnerabilities in open-source software via hypocrite commits," *Proc. Oakland, page to appear*, 2021.

[12] I. Haddad and B. Warner, "Understanding the open source development model," *Linux Journal*, 2011.

[13] R. Cox, "Our software dependency problem," *Unpublished essay, available online in January: https://research. swtch. com/deps. pdf*, 2019.

[14] J. Cappos, J. Samuel, S. Baker, and J. Hartman, "Package management security," 01 2008.

[15] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, "Towards measuring supply chain attacks on package managers for interpreted languages," *arXiv preprint arXiv:2002.01139*, 2020.

[16] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 995–1010, USENIX Association, Aug. 2019.

[17] A. Dann, H. Plate, B. Hermann, S. E. Ponta, and E. Bodden, "Identifying challenges for oss vulnerability scanners-a study & test suite," *IEEE Transactions on Software Engineering*, 2021.

[18] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman, "The matter of heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, (New York, NY, USA), p. 475–488, Association for Computing Machinery, 2014.

[19] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," 2020.

[20] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[21] S. Mauw and M. Oostdijk, "Foundations of attack trees," vol. 3935, pp. 186–198, 07 2006.

[22] M. Glassman and M. J. Kang, "Intelligence in the internet age: The emergence and evolution of open source intelligence (osint)," *Computers in Human Behavior*, vol. 28, no. 2, pp. 673–682, 2012.

[23] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

[24] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[25] K. Tuma, G. Calikli, and R. Scandariato, "Threat analysis of software systems: A systematic literature review," *Journal of Systems and Software*, vol. 144, pp. 275–294, 2018.

[26] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[27] C. S. Wright, *The IT regulatory and standards compliance handbook: How to survive information systems audit and assessments*. Elsevier, 2008.

[28] A. Blackstone *et al.*, "Principles of sociological inquiry: Qualitative and quantitative methods," 2018.

[29] W. Albert and T. Tullis, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013.

[30] L. A. Goodman, "Snowball Sampling," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148 – 170, 1961.

[31] J. Marjanović, N. Dalčeković, and G. Sladić, "Improving critical infrastructure protection by enhancing software acquisition process through blockchain," in *7th Conference on the Engineering of Computer Based Systems*, ECBS 2021, (New York, NY, USA), Association for Computing Machinery, 2021.

[32] 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*, CCS '21, (New York, NY, USA), p. 1755–1770, Association for Computing Machinery, 2021.

[33] R. Abdalkareem, V. Oda, S. Mujahid, and E. Shihab, "On the impact of using trivial packages: An empirical case study on npm and pypi," *Empirical Software Engineering*, vol. 25, no. 2, pp. 1168–1204, 2020.

[34] M. Ohm, A. Sykosch, and M. Meier, "Towards detection of software supply chain attacks by forensic artifacts," 08 2020.

[35] O. Duman, M. Ghafouri, M. Kassouf, R. Atallah, L. Wang, and M. Debbabi, "Modeling supply chain attacks in iec 61850 substations," in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6, 2019.

[36] K. Singi, V. Kaulgud, R. J. C. Bose, and S. Podder, "Shift - software identity framework for global software delivery," in *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*, pp. 122–128, 2019.

[37] A. V. Barabanov, A. S. Markov, M. I. Grishin, and V. L. Tsirlov, "Current taxonomy of information security threats in software development life cycle," in *2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–6, 2018.

[38] C.-A. Staicu, M. Pradel, and B. Livshits, "Synode: Understanding and automatically preventing injection attacks on node. js.," in *NDSS*, 2018.

[39] A. Sabetta and M. Bezzi, "A practical approach to the automatic classification of security-relevant commits," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 579–582, 2018.

[40] L. Neil, S. Mittal, and A. Joshi, "Mining threat intelligence about open-source projects and libraries from code repository issues and bug reports," in *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 7–12, 2018.

[41] 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*, ESEM '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[42] N. Vasilakis, B. Karel, N. Roessler, N. Dautenhahn, A. DeHon, and J. M. Smith, "Breakapp: Automated, flexible application compartmentalization.," in *NDSS*, 2018.

[43] N. Imtiaz, S. Thorn, and L. Williams, *A Comparative Study of Vulnerability Reporting by Software Composition Analysis Tools*. New York, NY, USA: Association for Computing Machinery, 2021.

[44] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Fourteenth symposium on usable privacy and security (SOUPS 2018)*, pp. 281–296, 2018.

[45] S. Benthall, "Assessing software supply chain risk using public data," in *2017 IEEE 28th Annual Software Technology Conference (STC)*, pp. 1–5, 2017.

[46] B. Pfretzschner and L. ben Othmane, "Identification of dependency-based attacks on node.js," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[47] B. A. Sabbagh and S. Kowalski, "A socio-technical framework for threat modeling a software supply chain," *IEEE Security Privacy*, vol. 13, no. 4, pp. 30–39, 2015.

[48] S. Zhang, X. Zhang, X. Ou, L. Chen, N. Edwards, and J. Jin, "Assessing attack surface with component-based package dependency," in *International Conference on Network and System Security*, pp. 405–417, Springer, 2015.

[49] H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 411–420, 2015.

[50] B. Delamore and R. K. L. Ko, "A global, empirical analysis of the shellshock vulnerability in web applications," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 1129–1135, 2015.

[51] S. Du, T. Lu, L. Zhao, B. Xu, X. Guo, and H. Yang, "Towards an analysis of software supply chain risk management," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2013.

[52] M. Silic and A. Back, "Information security and open source dual use security software: trust paradox," in *IFIP International Conference on Open Source Systems*, pp. 194–206, Springer, 2013.

[53] C. W. Axelrod, "Assuring software and hardware security and integrity throughout the supply chain," in *2011 IEEE International Conference on Technologies for Homeland Security (HST)*, pp. 62–68, 2011.

[54] 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*, pp. 1–8, 2011.

[55] P. R. Croll, "Supply chain risk management - understanding vulnerabilities in code you buy, build, or integrate," in *2011 IEEE International Systems Conference*, pp. 194–200, 2011.

[56] R. J. Ellison and C. Woody, "Supply-chain risk management: Incorporating security into software development," in *2010 43rd Hawaii International Conference on System Sciences*, pp. 1–10, 2010.

[57] G. Ferreira, L. Jia, J. Sunshine, and C. Kästner, "Containing malicious package updates in npm with a lightweight permission system," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1334–1346, 2021.

[58] A. Sejfia and M. Schäfer, "Practical automated detection of malicious npm packages," *arXiv preprint arXiv:2202.13953*, 2022.

[59] M. Balliauw, "Building a supply chain attack with .NET, NuGet, DNS, source generators, and more!," 2021. [Online; accessed 20-October-2021].

[60] L. Constantin, "Npm Attackers Sneak a Backdoor into Node.js Deployments through Dependencies," 2018. [Online; accessed 20-October-2021].

[61] C. Cimpanu, "17 Backdoored Docker Images Removed From Docker Hub," 2018. [Online; accessed 20-October-2021].

[62] "Plot to steal cryptocurrency foiled by the npm security team," 2019. [Online; accessed 20-October-2021].

[63] A. Sharma, "Inside the "fallguys" malware that steals your browsing data and gaming ims; continued attack on open source software."

[64] E. Roth, "Open source developer corrupts widely-used libraries, affecting tons of projects." https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected. [Accessed 15-Mar-2022].

[65] L. Tal, "Alert: peacenotwar module sabotages npm developers in the node-ipc package to protest the invasion of ukraine." https://snyk.io/blog/peacenotwar-malicious-npm-node-ipc-package-vulnerability/. [Accessed 18-Mar-2022].

[66] B. Kaplan and J. Qian, "A survey on common threats in npm and pypi registries," 2021.

[67] M. Taylor, R. K. Vaidya, D. Davidson, L. De Carli, and V. Rastogi, "Spellbound: Defending against package typosquatting," *arXiv preprint arXiv:2003.03471*, 2020.

[68] "PyPI Python repository hit by typosquatting sneak attack." https://nakedsecurity.sophos.com/2017/09/19/pypi-python-repository-hit-by-typosquatting-sneak-attack/. [Accessed 15-Mar-2022].

[69] "npm Blog Archive: 'crossenv' malware on the npm registry." https://blog.npmjs.org/post/163723642530/crossenv-malware-on-the-npm-registry. [Accessed 15-Mar-2022].

[70] "skcsirt-sa-20170909-pypi." https://www.nbu.gov.sk/skcsirt-sa-20170909-pypi/. [Accessed 15-Mar-2022].

[71] D.-L. Vu, "Typosquatting and combosquatting attacks on the python ecosystem," 07 2020.

[72] Bertus, "Discord Token Stealer Discovered in PyPI Repository." https://bertusk.medium.com/discord-token-stealer-discovered-in-pypi-repository-e65ed9c3de06. [Accessed 15-Mar-2022].

[73] R. Lakshmanan, "Malicious NPM Libraries Caught Installing Password Stealer and Ransomware." https://thehackernews.com/2021/10/malicious-npm-libraries-caught.html. [Accessed 15-Mar-2022].

[74] A. Bagmar, J. Wedgwood, D. Levin, and J. Purtilo, "I know what you imported last summer: A study of security threats in the python ecosystem," *CoRR*, vol. abs/2102.06301, 2021.

[75] N. P. Tschacher, *Typosquatting in programming language package managers*. PhD thesis, Universität Hamburg, Fachbereich Informatik, 2016.

[76] R. K. Vaidya, L. D. Carli, D. Davidson, and V. Rastogi, "Security issues in language-based sofware ecosystems," 2019.

[77] Bertus, "Cryptocurrency Clipboard Hijacker Discovered in PyPI Repository." https://bertusk.medium.com/cryptocurrency-clipboard-hijacker-discovered-in-pypi-repository-b66b8a534a8. [Accessed 15-Mar-2022].

[78] "Malicious packages found to be typo-squatting in Python Package Index." https://snyk.io/blog/malicious-packages-found-to-be-typo-squatting-in-pypi/. [Accessed 15-Mar-2022].

[79] "How to ask to ban the application for security reasons? Issue #651." https://github.com/canonical-web-and-design/snapcraft.io/issues/651. [Accessed 15-Mar-2022].

[80] D. Gonzalez, T. Zimmermann, P. Godefroid, and M. Schaefer, "Anomalicious: Automated detection of anomalous and potentially malicious commits on github," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 258–267, 2021.

[81] W. La Cholter, M. Elder, and A. Stalick, "Windows malware binaries in c/c++ github repositories: Prevalence and lessons learned.," in *ICISSP*, pp. 475–484, 2021.

[82] C. Paule, T. F. Düllmann, and A. Van Hoorn, "Vulnerabilities in continuous delivery pipelines? a case study," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 102–108, 2019.

[83] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 1393–1410, USENIX Association, Aug. 2019.

[84] A. Fass, M. Backes, and B. Stock, "Hidenoseek: Camouflaging malicious javascript in benign asts," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, (New York, NY, USA), p. 1899–1913, Association for Computing Machinery, 2019.

[85] S. Torres-Arias, A. K. Ammula, R. Curtmola, and J. Cappos, "On omitting commits and committing omissions: Preventing git metadata tampering that (re)introduces software vulnerabilities," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 379–395, USENIX Association, Aug. 2016.

[86] R. Goyal, G. Ferreira, C. Kästner, and J. Herbsleb, "Identifying unusual commits on github," *Journal of Software: Evolution and Process*, vol. 30, no. 1, p. e1893, 2018.

[87] L. Bass, R. Holz, P. Rimba, A. B. Tran, and L. Zhu, "Securing a deployment pipeline," in *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, pp. 4–7, 2015.

[88] "An emergency re-review of kernel commits authored by members of the university of minnesota, due to the hypocrite commits research paper." https://lore.kernel.org/lkml/202105051005.49BFABCE@keescook/. [Accessed 15-Mar-2022].

[89] J. Pewny and T. Holz, "Evilcoder: Automated bug insertion," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, (New York, NY, USA), p. 214–225, Association for Computing Machinery, 2016.

[90] "CVE - CVE-2021-42574." https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-42574. [Accessed 15-Mar-2022].

[91] "CVE - CVE-2021-42694." https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-42694. [Accessed 15-Mar-2022].

[92] "GitHub - mortenson/pr-sneaking: A repository demonstrating how you can sneak malicious code into Github PRs." https://github.com/mortenson/pr-sneaking. [Accessed 15-Mar-2022].

[93] "Why npm lockfiles can be a security blindspot for injecting malicious modules Snyk." https://snyk.io/blog/why-npm-lockfiles-can-be-a-security-blindspot-for-injecting-malicious-modules/. [Accessed 15-Mar-2022].

[94] H. Garrood, "Malicious code in the PureScript npm installer - Harry Garrood." https://harry.garrood.me/blog/malicious-code-in-purescript-npm-installer/. [Accessed 15-Mar-2022].

[95] "Trusted Relationship, Technique T1199 - MITRE att&ck." https://attack.mitre.org/techniques/T1199/. [Accessed 15-Mar-2022].

[96] "Valid Accounts, Technique T1078 - MITRE att&ck." https://attack.mitre.org/techniques/T1078/. [Accessed 15-Mar-2022].

[97] "Unsecured Credentials, Technique T1552 - MITRE att&ck." https://attack.mitre.org/techniques/T1552/. [Accessed 15-Mar-2022].

[98] "php.internals: Changes to Git commit workflow." https://news-web.php.net/php.internals/113838. [Accessed 15-Mar-2022].

[99] "Backdoor planted in PHP Git repository after server hack." https://portswigger.net/daily-swig/backdoor-planted-in-php-git-repository-after-server-hack. [Accessed 15-Mar-2022].

[100] "Malicious remote code execution backdoor discovered in the popular bootstrap-sass Ruby gem." https://snyk.io/blog/malicious-remote-code-execution-backdoor-discovered-in-the-popular-bootstrap-sass-ruby-gem/. [Accessed 15-Mar-2022].

[101] "strong_password v0.0.7 rubygem hijacked." https://withatwist.dev/strong-password-rubygem-hijacked.html. [Accessed 15-Mar-2022].

[102] "[CVE-2019-15224] Version 1.6.13 published with malicious backdoor. Issue #713." https://github.com/rest-client/rest-client/issues/713. [Accessed 15-Mar-2022].

[103] "Brute Force, Technique T1110 - MITRE att&ck." https://attack.mitre.org/techniques/T1110/. [Accessed 15-Mar-2022].

[104] E. Holmes, "How I gained commit access to Homebrew in 30 minutes." https://medium.com/@vesirin/how-i-gained-commit-access-to-homebrew-in-30-minutes-2ae314df03ab. [Accessed 15-Mar-2022].

[105] "CERT/CC Vulnerability Note VU#319816." https://www.kb.cert.org/vuls/id/319816. [Accessed 15-Mar-2022].

[106] "CAPEC - CAPEC-60: Reusing Session IDs (aka Session Replay) (Version 3.7)." https://capec.mitre.org/data/definitions/60.html. [Accessed 15-Mar-2022].

[107] N. Zahan, L. Williams, T. Zimmermann, P. Godefroid, B. Murphy, and C. Maddila, "What are weak links in the npm supply chain?," *arXiv preprint arXiv:2112.10165*, 2021.

[108] T. H. II, "Compromised npm Package: event-stream." https://medium.com/intrinsic-blog/compromised-npm-package-event-stream-d47d08605502. [Accessed 15-Mar-2022].

[109] T. K. Kuppusamy, V. Diaz, and J. Cappos, "Mercury: Bandwidth-effective prevention of rollback attacks against community repositories," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, (Santa Clara, CA), pp. 673–688, USENIX Association, July 2017.

[110] "OWASP Top Ten 2017 - A6:2017-Security Misconfiguration." https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration. [Accessed 15-Mar-2022].

[111] "CWE - CWE-16: Configuration (4.6)." https://cwe.mitre.org/data/definitions/16.html. [Accessed 15-Mar-2022].

[112] "Exploit Public-Facing Application, Technique T1190 - MITRE att&ck." https://attack.mitre.org/techniques/T1190/. [Accessed 15-Mar-2022].

[113] S. McClure, S. Gupta, C. Dooley, V. Zaytsev, X. B. Chen, K. Kaspersky, M. Spohn, and R. Permeh, "Protecting your critical assets-lessons learned from operation aurora," *Tech. Rep.*, 2010.

[114] "Operation ShadowHammer: A High Profile Supply Chain Attack." https://securelist.com/operation-shadowhammer-a-high-profile-supply-chain-attack/90380/. [Accessed 15-Mar-2022].

[115] C. Faulkner, "A hacker is demanding ransom for hundreds of stolen Git code repositories," 2019. [Online; accessed 20-October-2021].

[116] A. Warner (Gentoo Foundation) and A. K. Hüttel (Gentoo Foundation), "Project:Infrastructure/Incident Reports/2018-06-28 Github," 2018. [Online; accessed 20-October-2021].

[117] R. Naraine, "Open-source ProFTPD hacked, backdoor planted in source code." https://www.zdnet.com/article/open-source-proftpd-hacked-backdoor-planted-in-source-code/. [Accessed 15-Mar-2022].

[118] W. Bowling, "Git flag injection - local file overwrite to remote code execution," 2019. [Online; accessed 20-October-2021].

[119] P. A. Karger and R. R. Schell, "Thirty years later: Lessons from the multics security evaluation," in *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, pp. 119–126, IEEE, 2002.

[120] D. A. Wheeler, "Countering trusting trust through diverse double-compiling," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, pp. 13–pp, IEEE, 2005.

[121] P. Goswami, S. Gupta, Z. Li, N. Meng, and D. Yao, "Investigating the reproducibility of npm packages," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 677–681, 2020.

[122] D. L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, *Towards Using Source Code Repositories to Identify Software Supply Chain Attacks*, p. 2093–2095. New York, NY, USA: Association for Computing Machinery, 2020.

[123] V. Gruhn, C. Hannebauer, and C. John, "Security of public continuous integration services," in *Proceedings of the 9th International Symposium on Open Collaboration*, WikiSym '13, (New York, NY, USA), Association for Computing Machinery, 2013.

[124] W. Goerigk, "On trojan horses in compiler implementations," in *Proc. des Workshops Sicherheit und Zuverlassigkeit softwarebasierter Systeme*, Citeseer, 1999.

[125] A. Muñoz, "The octopus scanner malware: Attacking the open source supply chain," 2020.

[126] J. K. Smith, "Security incident on fedora infrastructure on 23 jan 2011," 2011.

[127] J. Rossignol, "What you need to know about ios malware xcodeghost," *URL: www. macrumors. com/2015/09/20/xcodeghost-chinese-malwarefaq*, 2015.

[128] "Adobe to revoke crypto key abused to sign malware apps (corrected)." https://arstechnica.com/information-technology/2012/09/adobe-to-revoke-crypto-key-abused-to-sign-5000-malware-apps/. [Accessed 15-Mar-2022].

[129] Webmin, "Webmin 1.890 exploit - what happened?," 2019. [Online; accessed 20-October-2021].

[130] "Siloscape: First Known Malware Targeting Windows Containers to Compromise Cloud Environments." https://unit42.paloaltonetworks.com/siloscape/. [Accessed 15-Mar-2022].

[131] F. Gröbert, A.-R. Sadeghi, and M. Winandy, "Software distribution as a malware infection vector," in *2009 International Conference for Internet Technology and Secured Transactions, (ICITST)*, pp. 1–6, 2009.

[132] D.-L. Vu, F. Massacci, I. Pashchenko, H. Plate, and A. Sabetta, "Lastpymile: Identifying the discrepancy between sources and packages," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, (New York, NY, USA), p. 780–792, Association for Computing Machinery, 2021.

[133] D.-L. Vu, "Py2src: Towards the automatic (and reliable) identification of sources for pypi package," *Statistics*, vol. 95, p. 27.

[134] K. Garrett, G. P. Ferreira, L. Jia, J. Sunshine, and C. Kästner, "Detecting suspicious package updates," *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp. 13–16, 2019.

[135] T. K. Kuppusamy, S. Torres-Arias, V. Diaz, and J. Cappos, "Diplomat: Using delegations to protect community repositories," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, (USA), p. 567–581, USENIX Association, 2016.

[136] X. de Carné de Carnavalet and M. Mannan, "Challenges and implications of verifiable builds for security-critical open-source software," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, (New York, NY, USA), p. 16–25, Association for Computing Machinery, 2014.

[137] J. Tellnes, "Dependencies: No software is an island," Master's thesis, The University of Bergen, 2013.

[138] K. Alhamed, M. C. Silaghi, I. Hussien, and Y. Yang, "Security by decentralized certification of automatic-updates for open source software controlled by volunteers," in *Workshop on Decentralized Coordination*, Citeseer, 2013.

[139] A. Ojamaa and K. Düüna, "Assessing the security of node.js platform," in *2012 International Conference for Internet Technology and Secured Transactions*, pp. 348–355, 2012.

[140] A. Bellissimo, J. Burgess, and K. Fu, "Secure software updates: Disappointments and new challenges.," in *HotSec*, 2006.

[141] M. Naedele and T. E. Koch, "Trust and tamper-proof software delivery," in *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, SESS '06, (New York, NY, USA), p. 51–58, Association for Computing Machinery, 2006.

[142] E. Levy, "Poisoning the software supply chain," *IEEE Security Privacy*, vol. 1, no. 3, pp. 70–73, 2003.

[143] J. Coelho, M. T. Valente, L. L. Silva, and E. Shihab, "Identifying unmaintained projects in github," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[144] A. Avram, "Npm was broken for 2.5 hours," *URL: www. infoq. com/news/2016/03/npm/*, 2016.

[145] "kik, left-pad, and npm," *URL: blog. npmjs. org/post/141577284765/kik-left-pad-and-npm*, 2016.

[146] M. Beltov, "Arch linux aur repository found to contain malware," *URL: sensorstechforum. com/arch-linux-aur-repository-found-contain-malware/*, 2018.

[147] A. Almubayed, "Practical approach to automate the discovery and eradication of opensource software vulnerabilities at scale," *Blackhat USA*, 2019.

[148] "Max.Computer - How to take over the computer of any Java (or Clojure or Scala) developer." https://max.computer/blog/how-to-take-over-the-computer-of-any-java-or-clojure-or-scala-developer/. [Accessed 16-Mar-2022].

[149] "CAPEC-142: DNS Cache Poisoning (Version 3.7)." https://capec.mitre.org/data/definitions/142.html. [Accessed 16-Mar-2022].
[150] "Attack inception: Compromised supply chain within a supply chain poses new risks." https://www.microsoft.com/security/blog/2018/07/26/attack-inception-compromised-supply-chain-within-a-supply-chain-poses-new-risks/. [Accessed 16-Mar-2022].
[151] "CWE - CWE-601: URL Redirection to Untrusted Site." https://cwe.mitre.org/data/definitions/601.html. [Accessed 16-Mar-2022].
[152] A. Birsan, "Dependency confusion: How i hacked into apple, microsoft and dozens of other companies," *URL: medium.com/@alex.birsan/dependency-confusion-4a5d60fec610*, 2021.
[153] "A Confusing Dependency." https://autsoft.net/hu/a-confusing-dependency/. [Accessed 16-Mar-2022].
[154] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, "A look in the mirror: Attacks on package managers," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, (New York, NY, USA), p. 565–574, Association for Computing Machinery, 2008.
[155] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, (New York, NY, USA), p. 61–72, Association for Computing Machinery, 2010.
[156] A. Even-Zohar, "Eslint: Compromising the build using supply chain attack," *URL: cycode.com/blog/eslint-compromising-the-build-using-supply-chain-attack/*, 2021.
[157] A. L. Johnson, "Dragonfly: Western energy companies under sabotage threat." https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=7382dce7-0260-4782-84cc-890971ed3f17&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments, 2014. [Accessed 16-Mar-2022].
[158] "Beware of hacked ISOs if you downloaded Linux Mint on February 20th!." https://blog.linuxmint.com/?p=2994. [Accessed 16-Mar-2022].
[159] "How to Bury a Major Breach Notification." https://krebsonsecurity.com/2017/02/how-to-bury-a-major-breach-notification/. [Accessed 16-Mar-2022].
[160] T. Chauchefoin, "Php supply chain attack on composer," *URL: blog.sonarsource.com/php-supply-chain-attack-on-composer*, 2021.
[161] M. Justicz, "Remote code execution on packagist.org," 2018. [Online; accessed 20-October-2021].
[162] M. Justicz, "Remote code execution on rubygems.org," 2017. [Online; accessed 20-October-2021].
[163] "Supply-chain attack on cryptocurrency exchange gate.io." https://www.welivesecurity.com/2018/11/06/supply-chain-attack-cryptocurrency-exchange-gate-io/. [Accessed 16-Mar-2022].
[164] A. Sharma, "Trick or treat: that 'twilio-npm' package is brandjacking malware in disguise!," *URL: blog.sonatype.com/twilio-npm-is-brandjacking-malware-in-disguise*, 2021.
[165] J. S. Meyers and B. Tozer, "Bewear! python typosquatting is about more than typos," 2020. [Online; accessed 20-October-2021].
[166] S. Mortenson, "pr-sneaking," 2017. [Online; accessed 20-October-2021].
[167] L. Giovanini, D. Oliveira, H. Sanchez, and D. Shands, "Leveraging team dynamics to predict open-source software projects' susceptibility to social engineering attacks," 2021.
[168] T. H. II, "Compromised npm Package: event-stream," 2018. [Online; accessed 20-October-2021].
[169] T. Costa, "strong_password v0.0.7 rubygem hijacked," 2019. [Online; accessed 20-October-2021].
[170] R. Naraine, "Open-source ProFTPD hacked, backdoor planted in source code," 2010. [Online; accessed 20-October-2021].
[171] S. J. Vaughan-Nichols, "Php supply chain attack shows open source's virtues and vices," 2021. [Online; accessed 20-October-2021].
[172] R. Lakshmanan, "Critical jenkins server vulnerability could leak sensitive information," 2020. [Online; accessed 20-October-2021].
[173] Microsoft Defender Security Research Team, "Attack inception: Compromised supply chain within a supply chain poses new risks," 2018. [Online; accessed 20-October-2021].
[174] Autosoft, "A confusing dependency," 2021. [Online; accessed 20-October-2021].
[175] J. Engelberg, "Bash uploader security update," *URL: about.codecov.io/security-update/*, 2021.
[176] M. Hanley, "Github's commitment to npm ecosystem security," 2021. [Online; accessed 20-October-2021].
[177] Google, "Google's open source documentation." [Online; accessed 20-October-2021].
[178] Microsoft Corporation, "3 ways to mitigate risk when using private package feeds," *URL: azure.microsoft.com/en-us/resources/3-ways-to-mitigate-risk-using-private-package-feeds/*, 2021.
[179] T. Durieux, C. Le Goues, M. Hilton, and R. Abreu, "Empirical study of restarted and flaky builds on travis ci," *Proceedings of the 17th International Conference on Mining Software Repositories*, Jun 2020.
[180] A. Kjäll, S. Kristoffersen, and S. Pettersen, "How we protected ourselves from the dependency confusion attack," *URL: schibsted.com/blog/dependency-confusion-how-we-protected-ourselves/*, 2021.
[181] C. Soto-Valero, N. Harrand, M. Monperrus, and B. Baudry, "A comprehensive study of bloated dependencies in the maven ecosystem," *Empirical Software Engineering*, vol. 26, Mar 2021.
[182] M. Taylor, R. Vaidya, D. Davidson, L. De Carli, and V. Rastogi, *Defending Against Package Typosquatting*, pp. 112–131. 12 2020.
[183] N. Forsgren, B. Alberts, K. Backhouse, G. Baker, G. Cecarelli, D. Jedamski, S. Kelly, and C. Sullivan, "2020 state of the octoverse: Securing the world's software," 2021.
[184] S. Du, T. Lu, L. Zhao, B. Xu, X. Guo, and H. Yang, "Towards an analysis of software supply chain risk management," 2013.
[185] B. Lamowski, C. Weinhold, A. Lackorzynski, and H. Härtig, "Sandcrust: Automatic sandboxing of unsafe components in rust," in *Proceedings of the 9th Workshop on Programming Languages and Operating Systems*, PLOS'17, (New York, NY, USA), p. 51–57, Association for Computing Machinery, 2017.
[186] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, p. 0–0, 2021.
[187] M. Ohm, L. Kempf, F. Boes, and M. Meier, "Supporting the detection of software supply chain attacks through unsupervised signature generation," *arXiv preprint arXiv:2011.02235*, 2020.
[188] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, (New York, NY, USA), p. 181–191, Association for Computing Machinery, 2018.
[189] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, pp. 297–300, 2010.
[190] GitHub, "The 2021 state of the octoverse," 2021. [Online; accessed 20-October-2021].
[191] R. Goyal, G. Ferreira, C. Kästner, and J. Herbsleb, "Identifying unusual commits on github: Goyal et al .," *Journal of Software: Evolution and Process*, vol. 30, p. e1893, 09 2017.
[192] D. Wheeler, "Countering trusting trust through diverse double-compiling," in *21st Annual Computer Security Applications Conference (ACSAC'05)*, pp. 13 pp.–48, 2005.
[193] European Network and Information Security Agency, "Enisa threat landscape for supply chain attacks 2021," 2021. [Online; accessed 20-October-2021].
[194] M. Ensor and D. Stevens, "Shifting left on security: Securing software supply chains," *URL: cloud.google.com/files/shifting-left-on-security.pdf*, 2021.

## APPENDIX A
## SAFEGUARDS AGAINST OSS SUPPLY CHAIN ATTACKS

Table II shows the identified safeguards mitigating attacks on OSS Supply Chain.

## APPENDIX B
## SCIENTIFIC AND GREY LITERATURE RESOURCES

The four digital libraries queried during the SLR are: Google Scholar[6], arXiv[7], IEEExplore[8] and ACM Digital Library[9]. The main sources used during the grey literature review are the following:

- IQT Lab's Software Supply Chain Compromises dataset [10];

[6]https://scholar.google.com/
[7]https://arxiv.org/
[8]https://ieeexplore.ieee.org/
[9]https://dl.acm.org/
[10]https://github.com/IQTLabs/software-supply-chain-compromises

| Safeguard | Control Type | | | | Stakeholders Involved | | | Attack-Vector Addressed |
|---|---|---|---|---|---|---|---|---|
| | *Directive* | *Preventive* | *Defective* | *Corrective* | *OSS Maintainer* | *3P Service Prov.* | *OSS Consumer* | |
| Maintain detailed SBOM and perform SCA | | ✓ | ✓ | | • | • | • | AV-000 |
| Code signing | | | ✓ | | • | • | • | AV-200, AV-500 |
| Use of security, quality and health metrics | ✓ | ✓ | | | • | • | • | AV-000 |
| Reproducible builds | | | ✓ | | • | • | • | AV-400, AV-500 |
| Secure authentication (e.g., MFA, password recycle, session timeout, token protection) | | ✓ | | | • | • | | AV-*00 → AV-602 |
| User account management | | ✓ | | ✓ | • | • | | AV-302,AV-402,AV-504,AV-600 |
| Audit | ✓ | | ✓ | | • | • | | AV-000 |
| Security assessment | | | ✓ | | • | • | | AV-000 |
| Vulnerability assessment | | | ✓ | | • | • | | AV-000 |
| Penetration testing | | | ✓ | | • | • | | AV-000 |
| Scoped packages | | ✓ | | | • | • | | AV-509 |
| Preventive squatting the released packages | | ✓ | | | • | • | | AV-200 |
| Pull/Merge request review | | ✓ | | | • | | | AV-301, AV-302 |
| Protect production branch | | ✓ | ✓ | | • | | | AV-301, AV-302 |
| Isolation of build steps | | ✓ | | | • | | | AV-400 |
| Ephemeral build environment | | ✓ | | | • | | | AV-400 |
| Use minimal set of trusted build dependencies in the release job | | ✓ | | | • | | | AV-400 |
| Restrict access to system resources of code executed during each build steps | | ✓ | | | • | | | AV-400 |
| Use of dedicated build service | | ✓ | | | • | | | AV-400 → AV-700 |
| Manual source code review | | | ✓ | | | • | • | AV-300 |
| Application Security Testing | | | ✓ | | | • | • | AV-000 |
| Build dependencies from sources | | ✓ | | | | • | • | AV-400, AV-500 |
| Typo guard/Typo detection | | ✓ | ✓ | | | • | • | AV-200 |
| Establish vetting process for Open-Source components hosted in internal/public repositories | | ✓ | | | | • | • | AV-000 |
| Runtime Application Self-Protection (RASP) | | | ✓ | ✓ | | | • | AV-000 |
| Remove un-used dependencies | | ✓ | | | | | • | AV-001 |
| Prevent script execution | | ✓ | | | | | • | AV-000 |
| Code isolation and sandboxing | | | | ✓ | | | • | AV-000 |
| Version pinning | | ✓ | | | | | • | AV-001 |
| Dependency resolution rules | | ✓ | | | | | • | AV-501, AV-508, AV-509 |
| Establish internal repository mirrors and reference one private feed, not multiple | | ✓ | | | | | • | AV-501,AV-502, AV-504, AV-505 |
| Integrate Open-Source vulnerability scanner into CI/CD pipeline | | | ✓ | | | | • | AV-000 |
| Integrity check of dependencies through cryptographic hashes | | | ✓ | | | | • | AV-400, AV-500 |

TABLE II: Safeguards against OSS supply chain attacks, incl. control type, stakeholder(s) involved in their implementation, and a mapping to mitigated attack vectors (cf. Figure 4 to resolve their identifiers).

- Backstabber's Knife Dataset [11] [19];
- Whitepapers from Microsoft [178] and Google [194];
- Whitepapers of projects for securing the software supply chain, like Supply-chain Levels for Software Artifacts (SLSA) [9], sigstore [12], TUF [13], in-toto [14] and OSSF Scorecard [15];
- News aggregator (e.g., The Hacker News [16], Bleepingcomputer [17], heise Security [18]);
- Blogs of package repositories and security vendors (e.g., Snyk [19], Sonatype [20]) and security researchers;
- Keynotes from cyber-security conferences (e.g., Blackhat [147]);
- MITRE's Common Attack Pattern Enumeration and Classification (CAPEC)[21];
- MITRE's ATT&CK[22].

[11] https://dasfreak.github.io/Backstabbers-Knife-Collection/
[12] https://www.sigstore.dev/
[13] https://theupdateframework.io
[14] https://in-toto.io
[15] https://github.com/ossf/scorecard
[16] https://thehackernews.com
[17] https://www.bleepingcomputer.com
[18] hhttps://www.heise.de/security/
[19] https://snyk.io/blog/
[20] https://blog.sonatype.com
[21] https://capec.mitre.org/
[22] https://attack.mitre.org