

Analyzing Challenges in Deployment of the SLSA Framework for Software Supply Chain Security

Mahzabin Tamanna

*North Carolina State University
Raleigh, NC, USA
mtamann@ncsu.edu*

Sivana Hamer

*North Carolina State University
Raleigh, NC, USA
sahamer@ncsu.edu*

Mindy Tran

*Max Planck Institute for Security and Privacy
Bochum, Germany
mindy.tran@mpi-sp.org*

Sascha Fahl

*CISPA Helmholtz Center for
Information Security, Hanover, Germany
sascha.fahl@cispa.de*

Yasemin Acar

*Paderborn University
The George Washington University
Paderborn, Germany and Wahington, USA
yasemin.acar@uni-paderborn.de*

Laurie Williams

*North Carolina State University
Raleigh, NC, USA
lawilli3@ncsu.edu*

Abstract—In 2023, Sonatype reported a 200% increase in software supply chain attacks, including major build infrastructure attacks. To secure the software supply chain, practitioners can follow security framework guidance like the Supply-chain Levels for Software Artifacts (SLSA). However, recent surveys and industry summits have shown that despite growing interest, the adoption of SLSA is not widespread. To understand adoption challenges, the goal of this study is to aid framework authors and practitioners in improving the adoption and development of Supply-Chain Levels for Software Artifacts (SLSA) through a qualitative study of SLSA-related issues on GitHub. We analyzed 1,523 SLSA-related issues extracted from 233 GitHub repositories. We conducted a topic-guided thematic analysis, leveraging the Latent Dirichlet Allocation (LDA) unsupervised machine learning algorithm, to explore the challenges of adopting SLSA and the strategies for overcoming these challenges. We identified four significant challenges and five suggested adoption strategies. The two main challenges reported are complex implementation and unclear communication, highlighting the difficulties in implementing and understanding the SLSA process across diverse ecosystems. The suggested strategies include streamlining provenance generation processes, improving the SLSA verification process, and providing specific and detailed documentation. Our findings indicate that some strategies can help mitigate multiple challenges, and some challenges need future research and tool enhancement.

Index Terms—SLSA, Software supply-chain security, Security Framework

I. INTRODUCTION

Software supply chain attacks are predicted to cost businesses \$138 billion globally by 2031 [1]. In 2023, Sonatype reported that 245,032 malicious open-source packages were downloaded, reflecting a 200% increase in software supply chain attacks [2]. Well-known build infrastructure attacks, like SolarWinds and Codecov, affected thousands of customers and hundreds of businesses and government agencies [3]. Consequently, the US Executive Order (EO) 14028 [4] highlights the urgent need to improve transparency and integrity in open-source artifacts to enhance supply chain security. The

Open Source Security Foundation (OpenSSF) introduced a security framework, Supply-chain Levels for Software Artifacts (SLSA) [5] (described in Sec. II). SLSA is designed to aid organizations in securing software and infrastructure development and deployment by preventing tampering and ensuring integrity. Organizations can leverage SLSA to implement artifact management practices and automate decisions to protect the software supply chain [6]–[8]. Software supply chain attacks are increasing. For example, in March 2024, xz-utils, a data compression library in Linux distributions, was compromised with malicious commits and introduced a backdoor affecting SSH as a dependency [9]. Implementing SLSA practices, such as source integrity verification, software provenance tracking, secure builds, and dependency verification, could help reduce the risk of software supply chain attacks. Several studies have highlighted the advantages of SLSA in securing software supply chain [10]–[12], but adopting SLSA poses challenges. Based on an industry summit and insights from 19 practitioners from 17 various companies, Tran et al. [13] reported that practitioners consider SLSA promising for securing systems but face limitations in integrating it into their ecosystems. Practitioners also have uncertainty about overcoming obstacles in securing buildings using SLSA. According to an analysis of the 90 projects from Java, Python, and npm ecosystems, most of the top projects across the ecosystems struggle to fulfill the core requirement of SLSA [14]. A joint survey conducted by the OpenSSF, Chainguard, Eclipse, and the Rust foundation in 2023 [15] revealed that over 50% of participants find it difficult to implement certain SLSA practices, such as hermetic builds. Moreover, survey respondents stated that making the provenance document available is a complex practice for organizations. Therefore, systematically analyzing the SLSA-related challenges and proposed strategies to overcome the challenges can help in increasing the adoption of security frameworks like SLSA.

The goal of this study to aid framework authors and practitioners in improving the adoption and development of Supply-Chain Levels for Software Artifacts (SLSA) through a qualitative study of SLSA-related issues on GitHub.

In this study, we address the following research questions:

- **RQ1:** What challenges do practitioners encounter while deploying SLSA?
- **RQ2:** What strategies do software practitioners suggest to framework authors for increasing SLSA adoption?

In this study, *framework authors* refers to those who developed or maintain the SLSA framework; *practitioners* refers to people who created issues for SLSA-related challenges, provided strategies to overcome them, or involved in the discussion on GitHub.

To address our research questions, we analyzed 1,523 SLSA-related issues from 233 GitHub repositories across June 2021 to September 2023. We used unsupervised machine learning, Latent Dirichlet Allocation (LDA), and qualitative analysis. Framework authors can improve framework usability and adoption based on our findings. Practitioners can gain an understanding of challenges and solutions to better secure their software supply chains. In addition, addressing these adoption challenges may enhance overall software ecosystem security, benefiting software development and deployment stakeholders. The contributions of our study are below:

- 1) A set of challenges practitioners encountered in adopting SLSA.
- 2) A set of strategies discussed by practitioners to overcome the challenges of deploying SLSA.
- 3) A set of recommendations based on our work for security framework authors, practitioners, and researchers to improve the adoption of security frameworks like SLSA.

II. SUPPLY-CHAIN LEVELS FOR SOFTWARE ARTIFACTS (SLSA)

Supply Chain Levels for Software Artifacts (SLSA) is an end-to-end security framework designed to enhance the integrity and security of software artifacts throughout the software supply chain. SLSA offers a checklist of standards and controls for preventing tampering, securing packages, and safeguarding the infrastructure involved in software development and distribution processes [5], [8], [16]. In 2021, a cross-industry collaboration led to the introduction of SLSA.

In April 2023, SLSA released v1.0, replacing v0.1 from 2021. The security requirements in SLSA are divided into tracks and levels. Each track of SLSA addresses specific aspects of supply chain security with defined requirements advancing to higher levels. While v0.1 had four tracks (source, build, provenance, common requirement) with levels 1 to 4, v1.0 focuses only on the build track with levels 0 to 3 (other tracks are deferred to future versions). The levels for both versions (0.1 and 1.0) are organized in ascending order and provide greater assurances of integrity as the level increases. SLSA v0.1 has four tracks; the *source track* focused on code protection, requiring a version-controlled repository

and trusted two-party review. The *build track* emphasized securing the artifact build, including requirements like build service and hermetic builds. The *provenance track* facilitated generating and consuming provenance, with requirements for availability, authentication, and non-falsifiable (now known as unforgeable). *Common requirements* applied to all trusted system components, covering security, access, and superuser needs.

SLSA is incorporated with tools `slsa-github-generator` [17] and `slsa-verifier` [18]. The `slsa-github-generator` assists GitHub projects in generating and managing SLSA provenance¹ by using GitHub actions. In addition, the CI/CD pipeline itself can also generate SLSA provenance as part of its build and deployment processes [19]. SLSA provenance is based on the in-toto [20] attestation framework, which creates a signed document that associates metadata with a software artifact. *Attestation* is an inspection process of verifying the authenticity and integrity of the generated provenance [5]. The `slsa-verifier` tool performs the verification process [18] and verifies the SLSA provenance generated by CI/CD builders by checking cryptographic signatures and matching expected values such as builder ID and source code repository.

For SLSA v1.0's build track, the requirements and objectives for each level are as follows:

- 1) No requirements and provides no artifact integrity guarantees. L0 represents the lack of SLSA.
- 2) Requires producers to provide automatically-generated provenance that shows how a project was built (e.g., used build processes, involved entities). Producers must distribute provenance to consumers, preferably through the respective package ecosystem.
- 3) Requires the build to run on a hosted platform that generates signed provenance. The downstream verification process involves validating the authenticity.
- 4) Producers need to implement strong security controls on their build platform so that build runs cannot influence one another even within the same project. Also, these controls should prevent any user-defined steps from accessing secret material used to sign the provenance.

III. METHODOLOGY

Our methodology comprises five steps: Platform Selection, Data Collection, LDA topic Modeling, Purposive Sampling, and Thematic Analysis. The steps in our methodology are discussed below and illustrated in Fig 1.

A. Data Collection Platform Selection

GitHub is a widely-used platform for software development [21] and scientific research [22]. Many research articles analyze data from GitHub [23], [24]. GitHub's open API and accessible features facilitate data collection and analysis, providing insights into Open-Source Software (OSS) development. SLSA hosts its project, documentation, and tools on

¹Provenance refers to metadata that provides information about how the outputs of a build were generated

GitHub. Authors and practitioners use GitHub issue trackers for questions, difficulties, clarifications, suggestions, and updates. Hence, we chose GitHub for data collection.

B. Data Collection

Our data collection from GitHub involved three phases: Search process, Data Cleaning and Selection, and Data Accuracy.

1) **Search Process:** We applied a multi-strategy approach to compile a comprehensive SLSA-related repositories: i)Any repository within the SLSA project, ii)Any repository that depended on `slsa-github-generator` based on the GitHub dependency graph [25], iii)Any repository using the `slsa-github-generator` and `slsa-verifier` tools in their GitHub workflow.

2) **Data Cleaning & Selection:** The search phase led to collecting 733 repositories. We removed duplicates, and 386 repositories remained. We reduced our initial dataset by 47%, primarily because our third search strategy involved gathering data when the GitHub action utilized the tools. Next, we filtered out repositories with no issues created, leaving 233 repositories collectively containing 56,992 issues. To refine our dataset, we systematically filtered these issues to isolate those directly pertinent to the SLSA framework. Since not all the issues gathered from the repositories referenced SLSA, we performed a keyword search using the specified search string to narrow our dataset. Our search string keyword search involves keywords including variations, contextual keywords, specific phrases, industry jargon, and iterative feedback. The first two authors discussed and refined the strings for relevance and accuracy. Our final search strings are: “Supply-chain Levels for Software Artifacts”, “SLSA”, “SLSA Framework”, “SLSA Security Framework”, “SLSA in Software Supply chain”, “OpenSSF SLSA”, “SLSA-verifier”, or “SLSA-generator”. We included issues that contained at least one of our selected search strings in the issue’s title, body, or comments. After this, 2,941 issues remained.

Next, we first discarded issues created by bots as we noticed bot-created issues were for automated tasks. Such bot-generated data introduces noise in communication [26]. We identified if a bot created an issue using GitHub GraphQL API and checked if the issue’s author type was a bot. Second, we excluded issues that contained terms indicative of automated tasks, such as branch pushing failures or scheduled workflow actions. For this, the first two authors collaboratively found some irrelevant automated issues based on any label or title included the keywords: “[e2e]”, “e2e:”, “e2e failure:”, or “cli:”, “[cli]”. Here, “e2e” and “cli” represent end-to-end testing and command line interface. We filtered issues based on these labels since no challenges or strategies-related discussions were reported. Filtering these issues was also necessary to avoid inaccurate interference of words in LDA-generated topics. After applying these filters, 1,523 issues were left for further analysis.

3) **Data Accuracy:** To validate our exclusion criteria, we randomly selected 20% of the excluded issues. The first author

manually reviewed the selected excluded issues to confirm their irrelevancy and inaccuracy with the desired dataset, which bolsters our method. We used the final dataset for LDA topic modeling.

C. LDA Topic Modeling

To extract underlying SLSA related topics from our dataset, we employed the Latent Dirichlet Allocation (LDA) [27]. LDA is a widely used probabilistic algorithm for topic modeling. LDA is suitable for identifying latent patterns in textual data, such as for our study of SLSA-related issues, which encompass a wide range of concerns and complexities in data. The lack of prior, in-depth research on SLSA underscores LDA’s application. LDA has been used in prior studies, such as NLP and sentiment analysis [28]–[31].

1) **Data Pre-processing:** The data pre-processing stage is essential to enhance the quality of unstructured text data analysis and improve human interpretability in LDA. Our data pre-processing included: i) removing punctuation, numbers, stop words, white spaces, and HTML tags, and converting all text to lowercase; ii) applying tokenization, where our content was divided into tokens (words), which were converted into a word vector; iii) applying text lemmatization to reduce words to their base; and iv) applying n-grams (bigrams and trigrams) to capture more data context. We observed some common words (GitHub, issue, slsa) and acronyms repeated several times within the topics. This repetition caused the duplication of words in the topics, which led to less distinctive keywords. As such, we removed these common words from the dataset, following prior work [31]. We also expanded general and domain-specific acronyms, for example, ‘BYOB’ to Build Your Own Builder and ‘sdlc’ to Software Development Life Cycle.

2) **Building Model:** To build the (LDA) model, we utilized MALLET [32], which employs Gibbs sampling algorithm [33]. We iterated the model with keyword counts from 5 to 20 and topics from 5 to 60, finding the ideal model at 50 topics with 10 keywords. We found the ideal model based on semantic consistency and the distribution of the topic-keywords based on discussion among the first two authors. Next, the first two authors labeled each topic separately based on the interpretation and the general meaning of the related keywords [34]. Then we finalized the labels and resolved conflicts by following a negotiation agreement practice [35] and mapped these topics into nine broader groups to facilitate the clustering of related topics [34]. The following nine labels were assigned as topic names and nine broader group names: documentation, terminology, provenance, attestation, workflow, defect management, version control, pertinence, and two-party review.

D. Purposive Sampling

For our qualitative analysis to address our research questions, we purposively sampled [36] issues from each of the nine broader groups of topics to conduct our inductive coding and thematic analysis. We opted for purposive sampling to

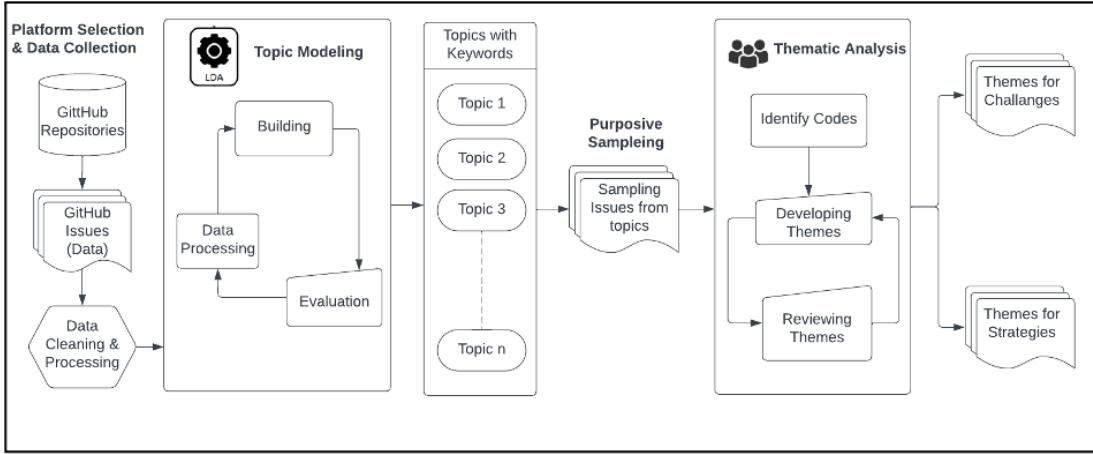


Fig. 1. Overview of Methodology

select our data, which aids in the chosen sample representing the population we are researching. We selected the top 20 issues from each group based on the composition value generated by (LDA). In the case of the “two-party review” group, we selected all 19 issues for our analysis because that group only contained 19 issues. As a result of our sampling process, we obtained 179 issues from 9 groups for qualitative coding.

E. Thematic Analysis

We performed a reflexive thematic analysis approach based on the six phases described by Braun and Clarke [37] on purposively sampled issues. We started with inductive coding [38] on selected issues for familiarization with the data and to avoid any biases toward current understanding. Inductive coding is widely utilized in academic research and involves analyzing data without a predetermined set of categories or themes [31], [39]. The first and second authors independently reviewed all selected issues and assigned initial codes. After the initial coding, the authors met to compare and discuss generated codes and collaboratively created the final codebook while resolving disagreements via discussion. We coded the issues from two aspects: (i) the challenges of adopting SLSA and (ii) strategies to overcome the challenges. Through an iterative process of comparing the codes, the first and second authors developed themes. Finally, to address the individual researcher positions inherent in qualitative research, such as our reflexive thematic analysis, we conducted group reflections with the first, second, and third authors on the identified themes [40]. We did not report inter-rater reliability, as this approach aligns with the principles of reflexive thematic analysis [41], and we resolved our conflicts as they emerged [42]. Also, the first author gathered supporting quotations for each theme. All the study material is available in GitHub repository [43].

For challenges: The process of theming challenges is based on the type and context of the difficulties. The step-by-step process is shown in Figure 2. For example, issues related to understanding SLSA-related terms were labeled as topic group

‘Terminology’ and grouped under sub-themes ‘Unclear Definitions’ Subsequently, the sub-theme ‘Unclear Definitions’ was grouped with the theme of ‘Unclear Communication (UC),’ which highlights difficulties in communicating the SLSA process. Table I represents the mapping of themes, sub-themes, and topic groups, along with the number of issues and sample issues.

For strategies: Practitioners proposed strategies to address SLSA challenges, which were categorized by type, context, and method. For example, strategies for improving SLSA documentation were grouped under ‘Enhance Documentation and Provide Patches.’ Strategies involving learning from negative experiences to improve documentation were categorized under ‘Use Negative Examples and Improve Design.’ These sub-themes were then grouped under the theme ‘Provide Specific and Detailed Documentation,’ emphasizing the need for better SLSA documentation.

IV. RESULTS

In this section, we present our findings for each research question. To address RQ1, we provide challenges encountered by practitioners in Section IV-A. In Section IV-B, we provide strategies practitioners suggested to overcome these challenges to answer RQ2. Additionally, we have included de-identified quotes from the developers and practitioners gathered from the GitHub issues related to the challenge themes.

A. Challenges

For *RQ1: What challenges do framework practitioners encounter while deploying SLSA?* we identified four themes of challenges: Complex Implementation (CI)(Section. IV-A1), Unclear Communication (UC)(Section. IV-A2), Limited Feasibility (LF)(Sec. IV-A3), and Unclear Relevancy (UR) (Section. IV-A4) from qualitative analysis. These four themes include subthemes. We also provided the total number of issues associated with each challenge theme in parentheses.

TABLE I
CATEGORIZATION OF THEMES AND SUB-THEMES OF TOPICS FOR CHALLENGES

Themes	Sub-theme	Topic groups	number of issues	number of sampled issues
CI. Complex Implementation	CI.1 Complicate Provenance Generation	Provenance	176	20
	CI.2 Intricate Maintenance	Workflow Defect Management Version Control	370 68 287	20 20 20
UC. Unclear Communication	UC.1 Unclear Definitions	Terminology	50	20
	UC.2 Unclear Documentation	Documentation	307	20
LF. Limited Feasibility	LF.1 Limited Attestation Verification	Attestation	200	20
	LF.2 Two-party Review Requirements	Two-party Review	19	19
UR. Unclear Relevance	UR.1 Unclear Relevance	Pertinence	46	20

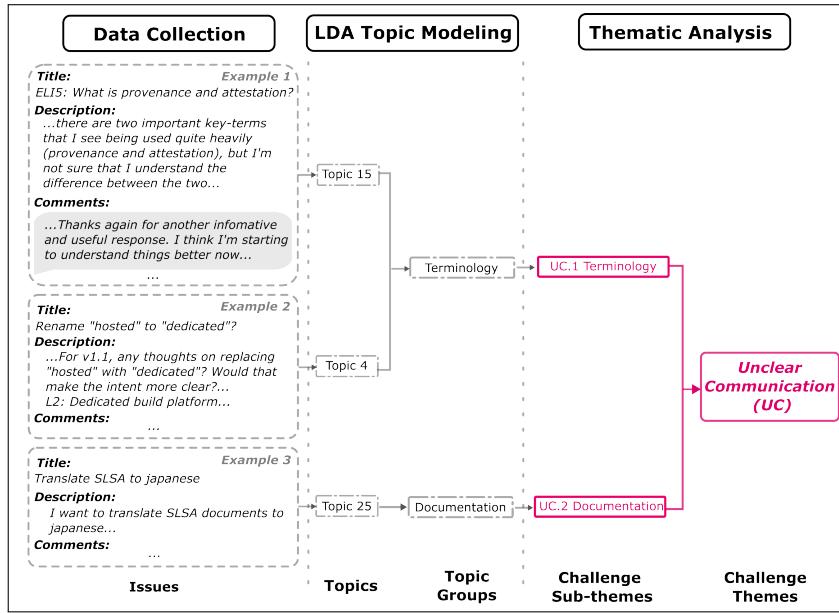


Fig. 2. Processing of issues to the challenge theme

1) **Complex Implementation (CI)(901):** CI is related to the challenges of integrating (SLSA) in projects. CI contains subthemes *Complicate Provenance Generation* and *Intricate Maintenance*.

CI.1 Complicated Provenance Generation: Practitioners reported challenges in generating provenance for higher levels of SLSA compliance. One major concern is the blocking nature and inefficiency of check-verifier pre-submit jobs, which can delay the submission process and affect the CI/CD pipeline's speed and agility. Another challenge is the lack of flexibility and support for "non-build" configurations in tools like `slsa-github-generator`. This inflexibility makes it difficult for practitioners with diverse use cases to fully utilize the tool. Additionally, some steps in using the `slsa-github-generator` can be laborious, especially with multiple builds or scripts, leading to uncertainty and difficulty in comprehension and incorporation. Security concerns also arise regarding storing sensitive information, such as usernames and login details, within the generated provenance. Ensuring secure handling of this information is important to prevent data breaches or unauthorized access. Furthermore, the

risk of missing information or data validity in the generated provenance impacts the verification process.

The integration of the slsa-framework action above is an educated guess from the instructions, but it runs. The artifact path, however, is a mystery: we've tried paths, relative paths, and the name of the generated container (tag) but are unsure exactly how to refer to the artifact

Standardization of the provenance generation process is essential to reduce confusion, emphasize consistency, and ensure data integrity and reliability in the build process. Semantic-release tool inconsistency in the python-package-template repository is an obstacle. The issue arises because the locally installed semantic-release tools may not match the ones used by the GitHub Action when running a CI workflow. Addressing these challenges requires technical expertise, process refinement, and adherence to security best practices throughout provenance generation and verification to meet higher levels of SLSA requirements.

CI.2 Intricate Maintenance: Practitioners stated maintaining the specifications of the required tools of SLSA is

complex. Running the `slsa-github-generator` and `slsa-github-verifier` tools causes various obstacles, including incompatibility, silent messages, runtime errors, hard coding, workflow-blocking steps, and data staleness. These technical challenges lead to confusion among users, who may be unsure about the expected behavior of the tools. Additionally, updates and modifications to the tools have sometimes resulted in discrepancies between the documentation and the actual code, complicating the maintenance process. Even minor adjustments can have unforeseen impacts, such as broken links, emphasizing the complex nature of managing project components.

Despite a successful login to the registry, there was an issue with the cosign attest process, indicating that the problem was not related to authentication or permissions. The user had to abandon a reusable workflow and manually verify the artifact's SHA before using it

Now it's really hard to see if tests are actually broken or are working fine just based on the workflow runs page. This is especially a problem because workflows fail "silently."

Integrating multiple tracks into SLSA has led to a debate on storing each track's levels. While detailed levels provide rich information, those can complicate adoption for users. Updating project tools for improvement can introduce new bugs that require remediation or backporting. Effectively managing these changes is imperative to ensure the project aligns with each version of SLSA.

2) **Unclear Communication (UC)(357):** UC is related to the challenges of understanding SLSA documentation. UC contains subthemes *Unclear Definitions* and *Unclear Documentation*.

UC.1 Unclear Definitions: Many practitioners expressed concerns about the lack of clarity in SLSA-related terminologies, emphasizing the importance of understanding these terms for system usability. Introducing less-known terms in the SLSA framework, such as "provenance" and "attestation," has created confusion among practitioners due to the absence of defined definitions and clear meanings. This unclarity has led to ambiguity and inaccuracies in the documentation, exemplified by terms like "hermetic build," "hosted," "build-service," and "non-falsifiable." Inconsistent terms in SLSA documentation created confusion, especially when used interchangeably or with unclear relationships. These discrepancies with standardization between SLSA and industry underscore the necessity for clear and standardized terminologies across various ecosystems and projects.

The words "resource" and "artifact" are only briefly explained at the beginning of the "Terminology" section & they aren't clear at all.

I was confused on the wording of "hosted" and asked in Slack.

However, practitioners stated that making conventional terms is not easy in industry due to their dynamic nature and can

vary based on context. Framework authors need to consider various factors while making the definitions as terminologies.

UC.2 Unclear Documentation: A common obstacle the practitioners encounter is the lack of clarity in explanation and guidelines on how to apply SLSA in their ecosystem. Practitioners expressed confusion about the level requirements and the usage of variables, such as whether variables like `(.Env.VERSION)` require prior definition using recommended code or if they are ready to use. Additionally, inconsistencies and unclear organization in document mapping have been highlighted as a challenge. For example, discrepancies in the documentation regarding ingesting a container workflow that does not align with the current functionality.

Need 'How to SLSA' for organizations and infrastructure providers

Following the documentation to migrate a project from GoReleaser, I was not able to understand that SLSA does not support "non-build" configurations

In addition, practitioners raised concerns regarding the website design of this framework. They expressed difficulty in navigating the pages. Specifically, the attack graphs are "confusing" to interpret. Practitioners also pointed out the lack of description of the process for merging Pull-Requests. Additionally, some practitioners expressed a lack of translation of the SLSA into various languages, such as German and Japanese, to increase its global adoption.

3) **Limited Feasibility (LF)(219):** LF is related to the challenges of the feasibility of the SLSA framework for improving the software supply-chain security. LF contains sub-themes *Limited Attestation Verification* and *Two-party Review Requirements*.

LF.1 Limited Attestation Verification: Provenance is at the core of the SLSA framework, aiding in documenting artifact authenticity. The verification process confirms artifacts' authenticity and integrity. However, practitioners are confused regarding attestation, such as its distinction from provenance and the necessity of automation in this process. Tools such as the `slsa-github-verifier` have been highlighted for complexity and redundancy, making the verification process less efficient. Moreover, a lack of clear guidance on how to communicate attestation data hinders downstream systems' ability to verify the data accurately.

One aspect of SLSA that's still in flux is where generated attestations should be stored. Basically, there isn't really a standardized way of doing this yet? And it sounds like you're implying that Sigstore is one candidate and storing attestations within an app's own repo is an option as well, though it sounded like you're saying whether or not to do that is still an "open question".

It is harder for systems to gain my trust from a producer's point of view

The diversity of environments for generating and storing data attestation adds flexibility but also introduces complex-

ties and potential documentation delays. Security concerns arise regarding the accuracy of attestation due to potential bugs or vulnerabilities in the verification process. Inconsistencies between package manager registries and actual files pose another risk, potentially undermining attestation accuracy. Additionally, not all signatures meet the integrity and authenticity requirements expected by the SLSA, limiting nuanced policy decisions. Establishing trust in supply chain sources is important to overcome these challenges.

LF.2 Two-party Review Requirements: The practice of two-party review checks for every change in software component and the code approved by two qualified reviewers. This practice helps to ensure that only trusted and authenticated authorized persons can make changes in software artifacts. However, practitioners encountered challenges in implementing this practice. Identifying suitable reviewers, assessing the practice's effectiveness, and applying it across various contexts proved to be complicated. Many open-source projects have only one maintainer or active user, making finding a second reviewer challenging. In addition, implementing such a review system presents an ongoing burden in terms of time and resources, raising concerns about its cost-effectiveness. Next, practitioners discussed inconsistencies in the security requirements between "Directly submit without review" and "Modify code after review" threats. Direct submissions require a review process, whereas modifications made after an initial review do not, which leads to potential security gaps and confusion among users.

It's unclear to me if pair programming and mob programming as acceptable instances of two trusted persons.

Concerns have emerged regarding the validity of pair programming, mob programming, and automated reviews as forms of two-person review. Practitioners questioned the security benefits of multiple reviews, highlighting the complexities of effective review processes.

4) **Unclear Relevance (UR) (46):** UR is related to the challenges of understanding the relevancy and significance of implementing SLSA. UR contains one sub-theme with the same name.

UR.1 Unclear Relevance: Practitioners faced confusion in identifying the specific attacks SLSA aims to mitigate and distinguishing SLSA from established security frameworks, standards, and ongoing projects. This lack of clarity makes it difficult to understand SLSA's unique benefits and value proposition. Additionally, developers struggled to determine how SLSA handles policies differently from OpenSSF best practices. For example, they were uncertain whether security policies should follow an OSSF-org policy or be managed by individual projects. This ambiguity further complicates practitioners' understanding of SLSA's distinct advantages.

Does any SLSA level help defend against Trojan horse attacks?

Aids organizations in creating an inventory of software and build systems used across a variety of

teams. Can we clarify this claim? How does it aid organizations? What does that mean?

Policy variations and inconsistency in SLSA hinder its effectiveness by causing compliance complexity, resource allocation issues, and process integration challenges. For example, a discrepancy exists between the npm registry and the package manager: installing with npm install P names the package A, while using npm download P followed by npm install P.tar.gz names it B. This inconsistency affects attestation, metadata, and provenance, persisting even if resolved at build time or with a lock file. Flexibility in incorporating SLSA is also crucial for enhancing testing procedures and security policies. Practitioners struggle to understand SLSA's effectiveness, particularly in creating a software repository for multiple teams, which hampers software management and collaboration.

B. Strategies

For RQ2:*What strategies do software practitioners suggest to framework authors for increasing SLSA adoption?*, we have analyzed strategies suggested by practitioners to overcome the challenges and identified five main themes and 13 sub-themes.

1) S1. Enhance SLSA alignment and flexibility:

S1.1 Incorporate Build-System Tracks: A "build systems track" should be included, focusing on the security of build systems in addition to the existing "build track" and "source track." Incorporating build system tracks within the SLSA framework allows for more tailored approaches to diverse system requirements. While current material on verifying build systems is a good start, it has limitations, such as the inability of people to self-assess a build system and the lack of visibility into how a provider performs.

S1.2 Gamify the Environment: Consumers want to know at a glance. Producers aim for higher levels as a form of gamification that can help them make their systems more secure. To provide rich information about the security of consumers, gamification can aid in promoting better adoption and comprehension of SLSA.

S1.3 Ensure Flexibility: Maintaining flexibility within the SLSA framework is essential to accommodate diverse system demands. The flexibility involves providing customizable and adaptable options based on specific organizational needs and evolving security landscapes. Furthermore, internal use cases can be different from what the open-source community needs.

S1.4 Align SLSA with OpenSSF Best Practices:

Practitioners suggested adding SECURITY.md, which at least points to the OpenSSF policy. The security policy should also identify the "security team" of members who are knowledgeable about security and will address security issues in order to better comply with OpenSSF security best practices. Adopting OpenSSF best practices silver and gold criteria, adding detailed reference docs for each action and workflow, and implementing an organization-wide security policy for the slsa-github-generator will enhance SLSA's effectiveness.

2) S2. Provide specific and detailed documentation:

S2.1 Enhance Documentation and Provide Patches: Improving the documentation involves providing clear and robust definitions, revising terms, using standard terms, and rewriting the get started page. The requirements need to be more precise and aligned with the SLSA levels to make it easy to apply them to any build system without any significant modifications.

S2.2 Use Negative Examples and Improve Diagram: Practitioners suggested the use of negative examples to explain complex concepts, define the framework's scope, or describe the purpose of requirements. This approach ensures an efficient and effective software management workflow while improving user experience and system functionality. Furthermore, practitioners have emphasized improving website design and incorporating diagrams to enhance user experience. For example, linking terminology pages with supply chain diagrams can provide a more precise understanding.

3) S3.Streamline Provenance Generation Processes:

S3.1 Simplify and Standardize Provenance Generation:

Standardizing the provenance generation process to enhance data integrity and reduce confusion and emphasizing consistency and reliability in builds are suggested. This approach simplifies and standardizes processes with tools and templates, making it easier for application owners to adopt provenance in development and deployment.

S3.2 Fix semantic-release tool inconsistency: To deal with semantic tool discrepancy in provenance generation, practitioners suggested defining clear rules for versioning based on semantic guidelines and using tools. Additionally, providing proper documentation and training to align teams and regularly test the release tool to catch issues early. Optimizing pre-submit jobs, such as running tasks in parallel, can also improve efficiency and consistency in releases.

4) S4.Improve SLSA Verification Process:

S4.1 Strengthen Verification Processes: To strengthen the verification process practitioners proposed enhancing security guarantees, providing algorithms for determining artifact levels, and offering additional evidence of verification. Implementing these strategies improves reliability, accuracy, and security in the verification.

S4.2 Implement Versioning Tagging: Practitioners emphasized the importance of implementing versioning tagging during the early stages of the SLSA framework to facilitate more straightforward tracking of progress and changes.

S4.3 Enhance SLSA Framework and Tool: Practitioners proposed enhancements to the SLSA framework and tool by adding more signaling information for downstream users. The strategy will improve the overall functionality and usability of SLSA.

5) S5.Collaborate with Community:

S5.1 Foster Community Engagement: Collaborating within communities aids in enhancing security measures [44]. Aligning SLSA verification practices with industry standards and guidelines promotes industry-wide compatibility and adoption. Providing clarification and explanations will aid novice users in understanding new technologies. Emphasizing community

engagement will help in enhancing framework usability, leading to improved user experiences.

S5.2 Promote Learning and Knowledge-Sharing:

Practitioners and framework authors have fostered a culture of learning and knowledge-sharing. Their collaboration is evident in addressing challenges like the two-party review, proposing solutions for single maintainers, low funding, and fewer requirements. For example, single maintainers can collaborate with other single-author projects for mutual support and expertise exchange.

V. DISCUSSION

We discuss the mapping between challenges and strategies and make recommendations for security framework authors (Section. V-A), practitioners (Section. V-B), and researchers (Section. V-C).

A. For security framework authors

Our findings highlight that Unclear Communication (UC) is a primary concern, leading to disinterest or confusion among adopters. To address the Unclear Definition challenge (UC.1), we recommend standardizing and consistently defining terminology, providing clear examples, and developing a comprehensive, easily accessible glossary. To deal with Unclear Documentation (UC.2), document quality can be improved by implementing strategies such as enhancing documentation and providing patches (S2.1), incorporating negative examples, improving diagrams (S2.2), increasing traceability, and considering translations. Framework authors should highlight the framework's uniqueness by contrasting it with other approaches and enhancing SLSA's alignment and flexibility (S1) to address the issue of Unclear Relevance (UR). Clear relevancy can be achieved by incorporating build-system tracks (S1.1), gamification of the environment (S1.2), ensuring flexibility (S1.3) and aligning SLSA with OpenSSF best practices (S1.4). These efforts collectively contribute to showcasing the framework's utility and increasing its appeal to potential adopters. Furthermore, Complex Implementation (CI) was found to be a significant challenge. To simplify this, the data suggests extending tools to improve the SLSA framework and tools (S4.2), fixing semantic-release tool inconsistency (S3.2) to deal with the discrepancy, simplifying and standardizing provenance generation (S3.1), and enhancing the SLSA framework and tool (S4.3).

We recommend framework authors enhance the SLSA framework by i) improving documentation by providing detailed guidelines, templates, and comprehensive examples; ii) implementing user-friendly strategies, such as designing intuitive interfaces and interactive demos; iii) automating processes with flexible tools by developing configurable tools for provenance generation and artifact verification, and integrate continuous security monitoring.

B. For practitioners

When adopting SLSA, practitioners should be aware of several challenges. First of all, understanding and ensuring that

the framework implementation aligns with the project’s security needs and goals. Due to concerns about Limited Feasibility (LF), practitioners should carefully study and verify SLSA’s security checks. For instance, verifying an attestation (LF.1) verifies specific build steps with particular inputs that lead to certain outputs. Weak links in certain package managers, such as those from npm, can potentially compromise the effectiveness of the attestation process [45]. Moreover, the two-party review (LF.2) might exclude certain projects when integrated into future framework versions. Practitioners should be aware of these limitations and plan their security strategies.

Despite the challenges, we recommend practitioners integrate SLSA and actively contribute to the improvement of the security framework to foster community engagement (S5.1) and promote learning and knowledge-sharing (S5.2). By engaging in collaborative efforts and sharing insights for enhancements, practitioners can play a vital role in evolving SLSA to address emerging challenges and strengthen its effectiveness across diverse software development environments. As such, always giving back to projects is vital for the sustainability of the ecosystem [46]. Practitioners can contribute by joining Slack to discuss with fellow developers, participating in community meetings, and improving SLSA with GitHub issues [47]. We recommend practitioners adopt the SLSA framework while carefully verifying security checks, understanding limitations, and ensuring its relevance to the project’s security needs. Actively contribute to the SLSA community by sharing insights, proposing enhancements, and participating in discussions to help evolve the framework and enhance its effectiveness.

C. For researchers

Several software supply chain challenges exist that require further research and development efforts. Practitioners often find it difficult to trust code that was not developed by themselves [48], which is essential in the correct generation of provenance (CI.1). Ensuring all provenance-building activities occur within the threat model’s trust boundary is important but complicated for users. Additionally, trust in the attestation (LF.1) is necessary. Inaccuracies, limitations, or vulnerabilities can harm the trust in the verification process and the actor. Practitioners mentioned possible exploits, including tampering with environments for pair programming, malicious collaborators, changing reviewed code, and subverting tools. To overcome the challenges, trust can be achieved by strengthening verification processes (S4.1) and implementing versioning tagging (S4.2). More research into mechanisms, tools, and standards to build trust throughout the supply chain is needed to increase the reliability of software packages [49]. Another challenge is the sustainability of open-source software projects when adopting SLSA practices. The two-party review process (LF.2) aims to balance security with the practicality of implementation for open-source, but many projects have few or single maintainers, making them susceptible to attacks. Two-party review can provide better guarantees but may limit accessibility and easy adaptability,

which is a key motivator for practitioners. Research is required on lowering adoption barriers, understanding contributor motivation, and mitigating disengagement. Studies can focus on detecting systems that are at risk through measurement [50], [51], easing adoption barriers for newcomers [52], addressing natural disengagement within projects [53] and understanding and improving developer motivation to contribute [54]. In addition, automating SLSA processes requires further work. While tools like `slsa-github-generator` and `slsa-github-verificator` aid adoption for provenance generation (CI.1) and verification (LF.1), more tooling is still needed.

VI. THREAT TO VALIDITY

In our study, we did not collect issues from other platforms, such as Reddit [55] and Stackoverflow [56]. We recognize the limitation by considering the widely adopted and recognized platform, Github, in software engineering and security studies [57]–[59]. Next, we did not collect demographic information such as gender, age, occupation, and technical background of practitioners. As such, we cannot assess the experience and perspectives of practitioners in the SLSA community. Moreover, this limitation restricted our capacity to generate descriptive statistics, such as tracking the number of issues created by individual users or investigating interactions between community members. We accounted for limitations in our topic modeling approach while optimizing models. Following Baumer et al. [60], we prioritized human-interpretable models, focusing on providing insightful data perspectives over seeking the optimal model. Our methodology, guided by LDA analysis practices [61], included thorough data cleaning and selecting topic numbers based on semantic coherence and keyword distribution. We also applied n-grams and word embedding to overcome the bag of word assumption of LDA. Finally, varying interpretations and potential oversights, manual analysis may introduce bias. For instance, the identified categories of challenges and strategies are susceptible to such bias. To address this, we cross-checked the identified categories and included only those on which both authors agreed.

VII. ETHICAL CONSIDERATION

Our Institutional Review Board (IRB) classified this study as “not human subjects” as we only utilized data that was publicly accessible and did not include interaction with humans. We also followed GitHub’s terms of use and guidelines for running academic research. According to GitHub’s Acceptable Use Policies [62]. We did not collect usernames or email addresses, so we will not publish any user information. We will publish only aggregate information and short, pseudonymized quotes from GitHub posts.

VIII. RELATED WORK

The software supply chain has become a frequently targeted attack vector in the field of cyberattacks [63]. For both open and closed-source supply chains, ensuring the reliable and efficient operation of their systems is essential [64]

as it involves compromising downstream dependents [65] and external factors [66] in different ecosystems [67]. Lella et al. [68] presented a taxonomy of supply chain attacks outlining the techniques used by attackers and the targeted assets. Ladisa et al. [69] classified attacks on open-source supply chains across all stages. Williams et al. [70] proposed the Proactive Software Supply Chain Risk Management (PSSCRM), a comprehensive framework for organizations to proactively manage software supply chain risks. Proposed solutions like The Update Framework (TUF) [71], in-toto [20] help to ensure supply chain integrity and secure distribution. In addition, tools and methods such as CDI [72], SPIRE [73], and Sigstore [74] provide software signing capabilities for developers, minimizing adoption barriers. Security tools protect package users by identifying and addressing known dependencies, but infrastructure is needed to build the framework. [10], [63], [75], [76]. Merala and Bowmen [10] stated, based on use case studies, that SLSA helps establish trust, enables trust flow between entities, and helps to ensure supply chain security. SLSA compliance with software bill of materials (SBOM) [77] to gain package information, SLSA can enhance software resiliency against potential attacks on the supply chain [78] [79]. According to Enck and Williams [80] findings from three industry summits, experts have a positive attitude towards SLSA, but securing the built environment remains challenging due to issues like trusting the compiler, among others. Organizations are adopting SLSA to fortify infrastructure, yet hurdles remain [13], [81]; drawbacks across software security frameworks lead to a lack of implementations [82]–[84] underscoring the necessity of addressing adoption challenges and enhancing the framework’s efficacy. Our research is inspired by previous studies, focusing on uncovering challenges faced by practitioners when deploying SLSA. We also explore strategies proposed by practitioners to help developers address challenges and improve SLSA adoption.

IX. CONCLUSION

Software security frameworks, such as SLSA, are designed to aid in securing projects throughout the software supply chain. However, despite the growing interest in adopting SLSA, practitioners face challenges. To understand the challenges and to find effective strategies to mitigate them, we conducted a content analysis of SLSA-related issues on GitHub. We analyzed 1,523 issues from 233 software repositories and leveraged probabilistic topic modeling (LDA) to identify latent topic sampling issues for qualitative analysis. Through thematic analysis, we identified four themes representing the challenges and five strategies to address them.

Our analysis revealed the top challenges, with the highest number of reported issues related to complex implementation and unclear communication of the SLSA process. The suggested strategies to address these challenges include streamlining provenance generation processes, improving the SLSA verification process, and providing specific and detailed documentation to overcome the challenges. Our findings em-

phasize the recurring need to simplify the implementation and understanding of security frameworks while enhancing trust in software supply chain security. Effective collaboration among framework authors, researchers, and practitioners is essential to improving adoption rates and strengthening software supply chain security.

ACKNOWLEDGMENT

This work was supported and funded by National Science Foundation Grant No. 2207008. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] “Software Supply Chain Attacks To Cost The World \$60 Billion By 2025,” <https://cybersecurityventures.com/software-supply-chain-attacks-to-cost-the-world-60-billion-by2025/>.
- [2] “Sonatype’s 9th Annual State of the Software Supply Chain Report Reveals Ways to Improve Developer, DevSecOps Efficiency,” <https://www.sonatype.com/press-releases/sonatype-9th-annual-state-of-the-software-supply-chain-report>.
- [3] Comparitech, “Worldwide software supply chain attacks tracker (updated daily),” <https://www.comparitech.com/software-supply-chain-attacks/>, Jul 2023.
- [4] “Executive Order on America’s Supply Chains,” <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/02/24/executive-order-on-americas-supply-chains/>.
- [5] Slsa, “Safeguarding artifact integrity across any software supply chain,” Dec 2023.
- [6] “What Is SLSA? SLSA Explained In 5 Minutes,” <https://www.legitsecurity.com/blog/what-is-slsa-slsa-explained-in-5-minutes>.
- [7] “SLSA: A Novel Framework For Secure Software Supply Chains,” <https://bytesafe.dev/posts/slsa-introduction/>.
- [8] K. Lewandowski and M. Lodato, “Introducing SLSA, an End-to-End Framework for Supply Chain Integrity,” <https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html>, June 2021.
- [9] A. Freund, “backdoor in upstream xz/liblzma leading to ssh server compromise,” <https://www.openwall.com/lists/oss-security/2024/03/29/4>, 2024.
- [10] M. S. Melara and M. Bowman, “What is software supply chain security?” *arXiv preprint arXiv:2209.04006*, 2022.
- [11] N. K. Tran, S. Pallewatta, and M. A. Babar, “Toward a reference architecture for software supply chain metadata management,” *arXiv preprint arXiv:2310.06300*, 2023.
- [12] P. Ladisa, S. E. Ponta, A. Sabetta, M. Martinez, and O. Barais, “Journey to the center of software supply chain attacks,” *IEEE Security & Privacy*, 2023.
- [13] M. Tran, Y. Acar, M. Cucker, W. Enck, A. Kapravelos, C. Kastner, and L. Williams, “S3c2 summit 2202-09: Industry secure supply chain summit,” *arXiv preprint arXiv:2307.15642*, 2023.
- [14] B. Hassanshahi, T. N. Mai, A. Michael, B. Selwyn-Smith, S. Bates, and P. Krishnan, “Macaron: A logic-based framework for software supply chain security assurance,” in *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2023, pp. 29–37.
- [15] (2023, 15 March) New SLSA++ Survey Reveals Real-World Developer Approaches to Software Supply Chain Security. <https://openssf.org/blog/2023/03/15/new-slsa-survey-reveals-real-world-developer-approaches-to-software-supply-chain-security/>.
- [16] B. Callaway, “Celebrating SLSA v1.0: securing the software supply chain for everyone,” <https://security.googleblog.com/2023/04/celebrating-slsa-v10-securing-software.html>, Apr 2023.
- [17] OpenSSF, “slsa-framework/slsa-github-generator,” <https://github.com/slsa-framework/slsa-github-generator>, May 2022.
- [18] ———, “slsa-framework/slsa-verifier,” <https://github.com/slsa-framework/slsa-verifier>, May 2022.

- [19] GitHub, “Using artifact attestations to establish provenance for builds.” [Online]. Available: <https://docs.github.com/en/actions/security-for-github-actions/using-artifact-attestations-using-artifact-attestations-to-establish-provenance-for-builds>
- [20] 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 *USENIX Security Symposium*, 2019, pp. 1393–1410.
- [21] C. Tozzi, “What Is GitHub and What Is It Used For?” <https://www.itprotoday.com/devops/what-github-and-what-it-used>, Sep 09 2022.
- [22] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, “A systematic mapping study of software development with github,” *Ieee access*, vol. 5, pp. 7173–7192, 2017.
- [23] N. Imtiaz, J. Middleton, J. Chakraborty, N. Robson, G. Bai, and E. Murphy-Hill, “Investigating the effects of gender bias on github,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 700–711.
- [24] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, “Categorizing the content of github readme files,” *Empirical Software Engineering*, vol. 24, pp. 1296–1327, 2019.
- [25] GitHub, “About the dependency graph,” Dec 2023. [Online]. Available: <https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph>
- [26] M. Wessel, B. M. De Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, “The power of bots: Characterizing and understanding bots in oss projects,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 2018.
- [27] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.
- [28] S. J. Uddin, M. Tamanna, A. Albert, and N. Pradhananga, “Leveraging natural language processing to identify health and safety challenges during post-disaster reconstruction,” in *Construction research congress 2022*, 2022, pp. 284–293.
- [29] S. J. Uddin, A. Albert, M. Tamanna, and A. Alsharef, “Youtube as a source of information: early coverage of the covid-19 pandemic in the context of the construction industry,” *Construction Management and Economics*, vol. 41, no. 5, pp. 402–427, 2023.
- [30] C. C. Silva, M. Galster, and F. Gilson, “Topic modeling in software engineering research,” *Empirical Software Engineering*, vol. 26, no. 6, p. 120, 2021.
- [31] R. P. Gauthier, M. J. Costello, and J. R. Wallace, “‘i will not drink with you today’: A topic-guided thematic analysis of addiction recovery on reddit,” in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–17.
- [32] A. K. McCallum, “Mallet,” <http://mallet.cs.umass.edu>, 2002.
- [33] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 721–741, 1984.
- [34] F. Gurcan and N. E. Cagiltay, “Big data software engineering: Analysis of knowledge domains and skill sets using lda-based topic modeling,” *Ieee access*, vol. 7, pp. 82 541–82 552, 2019.
- [35] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, “Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement,” *Sociological methods & research*, vol. 42, no. 3, pp. 294–320, 2013.
- [36] S. Campbell, M. Greenwood, S. Prior, T. Shearer, K. Walkem, S. Young, D. Bywaters, and K. Walker, “Purposive sampling: complex or simple? research case examples,” *Journal of research in Nursing*, vol. 25, no. 8, pp. 652–661, 2020.
- [37] V. Braun and V. Clarke, *Thematic analysis*. American Psychological Association, 2012.
- [38] Y. Chandra, L. Shang, Y. Chandra, and L. Shang, “Inductive coding,” *Qualitative research using R: A systematic approach*, pp. 91–106, 2019.
- [39] Y. Chandra and L. Shang, “Qualitative research using r: A systematic approach,” 2019.
- [40] J. W. Creswell and V. L. P. Clark, *Designing and conducting mixed methods research*. Sage publications, 2017.
- [41] V. Clarke and V. Braun, “Thematic analysis,” *The journal of positive psychology*, vol. 12, no. 3, pp. 297–298, 2017.
- [42] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, “Committed to trust: A qualitative study on security & trust in open source software projects,” in *2022 IEEE symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1880–1896.
- [43] S. on SLSA, “Using artifact attestations to establish provenance for builds.” [Online]. Available: <https://github.com/Mahzabin-Tamanna/Study-on-SLSA.git>
- [44] S. Amft, S. Höltervennhoff, R. Panskus, K. Marky, and S. Fahl, “Everyone for themselves? a qualitative study about individual security setups of open source software contributors,” in *In 45th IEEE Symposium on Security and Privacy (S&P’24)*, 2024.
- [45] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, “What are weak links in the npm supply chain?” in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 331–340.
- [46] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, “‘Always Contribute Back’: A Qualitative Study on Security Challenges of the Open Source Supply Chain,” in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1545–1560.
- [47] SLSA, “Community,” <https://slsa.dev/community>, Dec 2023.
- [48] L. Williams, “Trusting trust: Humans in the software supply chain loop,” *IEEE Security & Privacy*, vol. 20, no. 5, pp. 7–10, 2022.
- [49] M. Lieberman, “SLSA Is No Free Lunch,” <https://slsa.dev/blog/2022/04/slسا-is-no-free-lunch>, Apr 2022.
- [50] F. Calefato, M. A. Gerosa, G. Iaffaldano, F. Lanubile, and I. Steinmacher, “Will you come back to contribute? investigating the inactivity of oss core developers in github,” *Empirical Software Engineering*, vol. 27, no. 3, p. 76, 2022.
- [51] N. Zahan, P. Kanakiya, B. Hambleton, S. Shohan, and L. Williams, “Openssf scorecard: On the path toward ecosystem-wide automated security metrics,” *IEEE Security & Privacy*, 2023.
- [52] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, “Newcomers’ barriers... is that all? an analysis of mentors’ and newcomers’ barriers in oss projects,” *Computer Supported Cooperative Work (CSCW)*, vol. 27, pp. 679–714, 2018.
- [53] C. Miller, C. Kästner, and B. Vasilescu, “‘we feel like we’re winging it:’ a study on navigating open-source dependency abandonment,” in *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2023.
- [54] M. Gerosa, I. Wiese, B. Trinkenreich, G. Link, G. Robles, C. Treude, I. Steinmacher, and A. Sarma, “The shifting sands of motivation: Revisiting what drives contributors in open source,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1046–1058.
- [55] Reddit. [Online]. Available: <https://www.reddit.com/>
- [56] Stackoverflow. [Online]. Available: <https://stackoverflow.com/>
- [57] M. AlMarzouq, A. AlZaidan, and J. AlDallal, “Mining github for research and education: challenges and opportunities,” *International Journal of Web Information Systems*, vol. 16, no. 4, pp. 451–473, 2020.
- [58] E. REMY, “How pew research center uses git and github for version control,” Aug 2022. [Online]. Available: <https://www.pewresearch.org/decoded/2022/08/01/how-pew-research-center-uses-git-and-github-for-version-control/>
- [59] A. A. Kazmi, “Using github for academic research,” March 2024. [Online]. Available: https://hackmd.io/@vivek-blog/github_article
- [60] E. P. Baumer, D. Mimno, S. Guha, E. Quan, and G. K. Gay, “Comparing grounded theory and topic modeling: Extreme divergence or unlikely convergence?” *Journal of the Association for Information Science and Technology*, vol. 68, no. 6, pp. 1397–1410, 2017.
- [61] D. Maier, A. Waldherr, P. Miltner, G. Wiedemann, A. Niekler, A. Keinert, B. Pfetsch, G. Heyer, U. Reber, T. Häussler *et al.*, “Applying lda topic modeling in communication research: Toward a valid and reliable methodology,” in *Computational methods for communication science*. Routledge, 2021, pp. 13–38.
- [62] GitHub, “Github acceptable use policies (information usage restrictions).”
- [63] C. Liu, S. Chen, L. Fan, B. Chen, Y. Liu, and X. Peng, “Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 672–684.
- [64] E. Raymond, “The cathedral and the bazaar,” *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.
- [65] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s knife collection: A review of open source software supply chain attacks,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*:

- 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17.* Springer, 2020, pp. 23–43.
- [66] 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.
 - [67] 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)*, 2019, pp. 995–1010.
 - [68] I. Lella, M. Theocharidou, E. Tsekmezoglou, A. Malatras, and S. García, *ENISA Threat Landscape for Supply Chain Attacks*. ENISA, 2021.
 - [69] P. Ladisa, H. Plate, M. Martinez, and O. Barais, “Sok: Taxonomy of attacks on open-source software supply chains,” in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1509–1526.
 - [70] L. Williams, S. Miguez, J. Boote, and B. Hutchison, “Proactive software supply chain risk management framework (p-sscrm) version 1,” *arXiv preprint arXiv:2404.12300*, 2024.
 - [71] 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*, 2010, pp. 61–72.
 - [72] “Hardware-Enforced Integrity and Provenance for Distributed Code Deployments. In NIST Workshop on Enhancing Software Supply Chain Security,” Jul 2021.
 - [73] “The Rust Team. [n.d.], howpublished=https://clang.llvm.org/docs/DataFlowSanitizer.html, publisher=CDI, month=Jul, year=2022.”
 - [74] “sign, verify, protect.” 2023.
 - [75] E. Wyss, L. De Carli, and D. Davidson, “What the fork? finding hidden code clones in npm,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2415–2426.
 - [76] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, “Sok: Analysis of software supply chain security by establishing secure design properties,” in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022, pp. 15–24.
 - [77] “SPDX Overview,” <https://spdx.dev/about/overview/>, Jul 2023.
 - [78] “Google SLSA Cybersecurity Framework: Key Takeaways,” NOVEMBER 10 2021.
 - [79] “SPDX Overview.” <https://spdx.dev/about/>, retrieved Jul 2022.
 - [80] W. Enck and L. Williams, “Top five challenges in software supply chain security: Observations from 30 industry and government organizations,” *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.
 - [81] T. Dunlap, Y. Acar, M. Cucker, W. Enck, A. Kapravelos, C. Kastner, and L. Williams, “S3c2 summit 2023-02: Industry secure supply chain summit,” *arXiv preprint arXiv:2307_16557*, 2023.
 - [82] “How to implement 3 new software supply chain security frameworks,” August 30, 2021.
 - [83] K. G. Kalu, T. Singla, C. Okafor, S. Torres-Arias, and J. C. Davis, “An industry interview study of software signing for supply chain security,” *arXiv preprint arXiv:2406.08198*, 2024.
 - [84] C. E. M. Zottmann, T. M. S. do Amaral, R. R. Nunes, and J. J. C. Gondim, “Comparing software supply chain protection approaches,” in *2023 Workshop on Communication Networks and Power Systems (WCNPS)*. IEEE, 2023, pp. 1–7.