JACoIT
Journal of Applied Computer and Information Technology

Review

Penetration testing for software supply chain security: A lifecycle-oriented review of vulnerabilities, third-party dependencies, and mitigation strategies

Abdiqani Cusman Abdalla [1], Yuchong Cui [1,2,*] and Ihab Hussein Al Musawi [1]

[1]School of Computing, Universiti Utara Malaysia; abdiqanicusman9@gmail.com; hushobig@gmail.com
[2]Shandong Vocational and Technical University of Engineering, Shandong, China; cuiyuchong00@gmail.com

[*]Corresponding Author: cuiyuchong00@gmail.com

Received: 15 April 2026
Revised: 22 June 2026
Accepted: 27 June 2026
Published: 4 August 2026

Abstract

Modern software systems increasingly rely on open-source components, third-party libraries, cloud services, containers, and automated CI/CD pipelines. This dependence improves development speed but also expands the software supply chain attack surface beyond internally written code. Vulnerabilities may be introduced through transitive dependencies, build scripts, package repositories, artifact storage, or software update channels. This review examines how penetration testing can support software supply chain security by moving beyond passive vulnerability identification toward dynamic validation of exploitable risk. A structured review approach was used to examine recent literature on software supply chain attacks, SBOM adoption, dependency governance, CI/CD security, and security testing. The findings show that penetration testing is most useful when it is mapped to specific lifecycle stages, including dependency selection, development environments, build pipelines, artifact repositories, distribution mechanisms, and monitoring. The review also identifies practical limitations, such as unclear test boundaries, incomplete dependency visibility, limited automation, and resource constraints. The paper contributes a lifecycle-oriented view of penetration testing for software supply chain security and highlights how testing can complement SBOMs, secure development practices, and continuous monitoring.

Keywords: software supply chain security; penetration testing; supply chain attack; third-party dependency; SBOM; CI/CD pipeline

1. Introduction

Software supply chain security has become a central cybersecurity concern because modern software is assembled through open-source packages, third-party services, cloud infrastructure, container images, build tools, and automated CI/CD pipelines. This interconnected development model improves speed and reuse, but it also creates security

dependencies that extend beyond internally written code. Recent studies and standards show that risks may arise from external packages, transitive dependencies, build environments, artifact repositories, and update mechanisms rather than from application code alone [1]-[6].

Penetration testing is relevant to this context because supply chain attacks do not always appear as ordinary application vulnerabilities. Attackers may exploit package-source trust, manipulate dependency resolution, abuse CI/CD credentials, tamper with build scripts, replace artifacts, or compromise update channels. Conventional vulnerability scanning and SBOM generation improve visibility, but they do not always validate whether a weakness is reachable, exploitable, or controlled effectively [7]-[11].

Existing review papers have made important contributions by classifying software supply chain attacks, reviewing SBOM adoption, examining dependency risks, and discussing mitigation strategies [4]-[6], [9]. However, less attention has been given to how penetration testing can be mapped to specific lifecycle stages as a dynamic validation activity. This leaves a knowledge gap between identifying supply chain risks and testing whether those risks can be exploited in realistic development and delivery environments.

This review therefore focuses on the unresolved problem of penetration testing integration within software supply chain security. It asks how penetration testing can support exploitability validation across dependency selection, development environments, build pipelines, artifact repositories, distribution mechanisms, and monitoring. The lifecycle-oriented perspective adds value because it connects testing activities to concrete supply chain control points rather than treating penetration testing as a generic post-development activity.

The objectives of this review are to identify major software supply chain attack vectors, examine the role of penetration testing in validating supply chain risks, synthesize the main challenges that limit testing adoption, and develop a lifecycle-oriented synthesis model. The model is presented as a conceptual synthesis derived from reviewed literature, not as a fully validated operational framework.

2. Literature Review

The literature review situates this study within existing work on software supply chain security, software supply chain attacks, and penetration testing. Prior studies have mainly examined dependency risk, package ecosystems, SBOM adoption, CI/CD pipeline threats, and secure software development practices. However, fewer studies have directly explained how penetration testing can be adapted across the software supply chain lifecycle. Table 1 therefore positions the present review against closely related studies and clarifies its specific contribution.

Table 1. Positioning of this review against existing review directions

Review direction	Common focus	Remaining gap addressed here
Attack taxonomies	Classify attack vectors, dependency abuse, package compromise, and distribution attacks	Limited explanation of how penetration testing validates lifecycle-stage risk
SBOM and dependency studies	Improve component transparency and vulnerability response	Limited validation of reachability, exploitability, and control effectiveness
CI/CD security studies	Identify build-pipeline threats, secrets exposure, and workflow risks	Limited practical mapping to penetration testing activities
Risk management frameworks	Governance, supplier assessment, and secure development controls	Limited synthesis of testing evidence and remediation feedback

2.1. Software Supply Chain Security and Dependency Risk

Software supply chain security concerns the protection of the code, dependencies, build processes, tooling, infrastructure, and distribution mechanisms involved in software production. It differs from traditional application security because risks can originate from external ecosystems that the consuming organisation does not fully control [2], [5], [12]. Dependency risk is especially important because modern applications rely on direct and transitive packages whose provenance, maintenance status, and vulnerability exposure may be difficult to observe [13]-[15].

Prior studies show that dependency-related risks include outdated packages, bloated dependencies, vulnerable functions that are reachable only under specific conditions, and malicious packages distributed through public repositories [16]-[21]. These risks demonstrate why software supply chain security requires both visibility mechanisms and validation mechanisms.

2.2. Software Supply Chain Attacks and Lifecycle Exposure

Software supply chain attacks can occur at several stages of the software lifecycle. Dependency-based attacks exploit package managers, public repositories, dependency confusion, typosquatting, malicious maintainers, and transitive dependency chains [6], [14], [22]. Build-related attacks target CI/CD workflows, runners, scripts, credentials, and artifact generation processes [8], [23]. Distribution attacks may compromise signing, update delivery, or trusted release channels, as illustrated by SolarWinds [24].

Existing taxonomies provide a valuable foundation for understanding attack vectors [6], [25], [26]. Nevertheless, taxonomy alone does not show how organisations should test whether a particular supply chain control fails under realistic attack conditions. This limitation motivates the penetration-testing focus of the present review.

2.3. Penetration Testing in Software Supply Chain Contexts

Penetration testing traditionally evaluates exposed applications, network services, authentication controls, and misconfigurations. In software supply chain contexts, the test scope becomes broader. It may include dependency confusion simulation, repository permission testing, secret exposure checks, CI/CD workflow abuse, artifact replacement testing, signing control verification, and update-channel validation [3], [7], [8].

This role differentiates penetration testing from passive visibility mechanisms. SBOMs and dependency graphs can indicate what components exist, while penetration testing examines whether a path can be abused. This distinction is central to the novelty of the review: the paper synthesises penetration testing as an exploitability validation layer across the software supply chain lifecycle.

2.4. SBOMs, Standards, and Mitigation Frameworks

SBOMs, secure development standards, software supply chain risk management practices, and provenance frameworks are widely discussed as mechanisms for improving transparency and accountability [1], [2], [9], [10]. Empirical studies show that SBOM adoption is increasing, but SBOM completeness, accuracy, exploitability interpretation, and tool support remain practical challenges [9], [27].

Standards and frameworks provide important guidance, but they do not replace testing. The literature suggests that secure software development requires a combination of component visibility, build integrity, artifact provenance, continuous monitoring, and dynamic validation [1]-[3], [8], [28].

3. Materials and Methods

3.1. Review Protocol

This review adopted the SALSA logic of Search, Appraisal, Synthesis, and Analysis to improve methodological transparency and reproducibility. A PRISMA-style flow diagram was used to report the screening process, including record identification, duplicate removal, title and abstract screening, full-text assessment, and final study inclusion. Therefore, the study was positioned as a structured literature review rather than a purely narrative review.

The scope was limited to studies that directly inform software supply chain security, third-party dependency risk, SBOMs, CI/CD security, artifact integrity, secure software development, penetration testing, or security validation. The aim was not to measure attack frequency in industry, but to synthesize evidence on how penetration testing can support supply chain assurance.

3.2. Search Strategy and Data Sources

Searches were conducted in IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Taylor & Francis Online, and Google Scholar. Google Scholar was used only for supplementary checking, citation tracing, and DOI verification. The search focused on studies published between 2015 and 2026, with priority given to peer-reviewed sources from 2021 to 2026.

The search string combined three concept groups: software supply chain security, penetration/security testing, and lifecycle control points. Backward and forward citation checking was also used to identify additional peer-reviewed studies relevant to dependency risk, SBOM adoption, CI/CD pipeline security, and supply chain attacks.

Table 2. Search strategy and review scope

Element	Description
Databases	IEEE Xplore; ACM Digital Library; ScienceDirect; SpringerLink; Taylor & Francis Online; Google Scholar for supplementary checking and DOI verification
Search string	("software supply chain security" OR "software supply chain attack" OR "dependency vulnerability") AND ("penetration testing" OR "security testing" OR "dynamic testing") AND (SBOM OR CI/CD OR "build pipeline" OR "third-party dependency" OR "artifact repository")
Time frame	2015-2026, with priority on peer-reviewed work from 2021-2026
Final synthesis	34 sources retained after screening, quality appraisal, and relevance checks

3.3. Screening and Selection

The initial search identified 164 records. After 31 duplicates were removed, 133 records were screened by title and abstract. Sixty-eight records were excluded at this stage because they were not directly related to software supply chain security or security validation. Sixty-five full texts were assessed, and 31 were excluded with reasons. Thirty-four studies were retained for thematic synthesis. Records were excluded when they discussed general cybersecurity without software supply chain relevance, lacked a clear source, duplicated another included study, provided only vendor marketing material, or did not contribute technical, methodological, empirical, or conceptual evidence.

Table 3. Full-text exclusion reasons

Reason for Exclusion	Number Excluded
Not focused on software supply chain security	9
General cybersecurity article without testing or lifecycle relevance	7
Vendor marketing or unsupported opinion source	5
Duplicate topic already covered by stronger peer-reviewed source	6
Insufficient methodological, empirical, or technical contribution	4

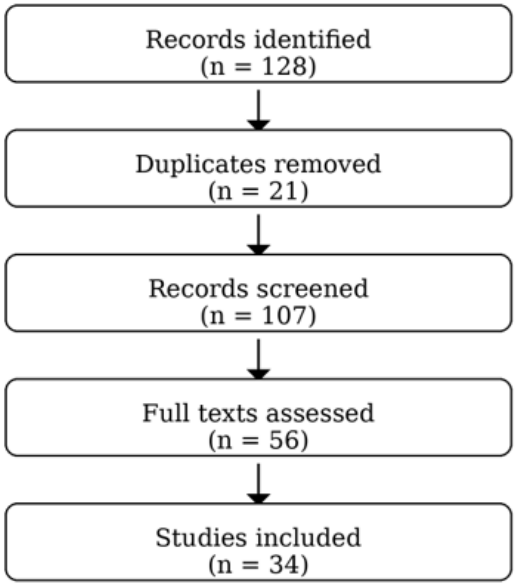

Figure 1. PRISMA-style study selection process used in the review

3.4. Quality Appraisal

Each candidate source was appraised for topic relevance, methodological clarity, source credibility, evidence strength, and usefulness for penetration testing or supply chain security. Peer-reviewed journal and conference papers were treated as core evidence. Standards and technical reports with DOI identifiers were used as supporting guidance. Preprints were used cautiously and only where they provided specific technical evidence not available in peer-reviewed form.

For scoring decisions, each criterion in Table 4 was rated from 0 to 2. A score of 0 indicated that the source did not meet the criterion, 1 indicated partial fulfilment, and 2 indicated clear fulfilment. Sources scoring 8-10 were treated as strong evidence, sources scoring 6-7 were retained only when they made a direct contribution to the review scope, and sources below 6 were excluded unless they were standards or technical reports required for contextual support.

To reduce single-reviewer bias, the appraisal and coding decisions were checked through author cross-review. One author conducted the initial screening and scoring, while another author verified borderline inclusion decisions, DOI traceability, source type, and thematic coding. Disagreements were resolved through discussion until consensus was reached. This process was used to improve consistency rather than to claim formal inter-rater reliability.

Table 4. Quality appraisal criteria

Quality Criterion	Score
Clear research aim or technical focus	0-2
Relevance to software supply chain security	0-2
Relevance to penetration/security testing or validation	0-2
Evidence strength or methodological clarity	0-2
Source credibility and DOI traceability	0-2
Total	10

3.5. Synthesis and Analysis

The selected studies were coded into recurring themes: dependency risk, CI/CD and build security, SBOM and component visibility, penetration testing adaptation, mitigation and governance, and case-based evidence. The analysis focused on identifying relationships between visibility mechanisms, exploitability validation, and lifecycle stages. This synthesis formed the basis for the findings and the conceptual lifecycle model presented in the Results and Discussion sections.

The final corpus of 34 sources was considered sufficient for the review purpose because the selection was guided by relevance, quality, and coverage of the lifecycle-oriented research question rather than by numerical breadth alone. The retained sources covered dependency risk, CI/CD and build security, SBOM visibility, penetration testing adaptation, mitigation and governance, and case evidence. Theme saturation was reached when additional candidate sources repeated existing concepts without adding distinct technical or analytical insight.

4. Results

The results section reports synthesis findings generated from the reviewed evidence rather than repeating background literature. The findings are organised around recurring themes identified during coding and quality appraisal. The analysis therefore goes beyond counting themes. It compares where the literature provides visibility, where it provides exploitability validation, and where lifecycle-stage testing remains underdeveloped. This distinction is important because software supply chain assurance requires evidence that controls work under realistic attack paths, not only evidence that risks have been listed.

Table 5. Thematic synthesis of reviewed evidence

Theme	Studies	Main issue identified	Implication for penetration testing
Third-party dependency risk	21	Transitive dependencies, outdated packages, malicious packages, and dependency bloat	Prioritise high-risk dependency paths and validate reachability
CI/CD and build security	17	Temporary runners, secrets, workflow scripts, and artifact generation	Embed lightweight tests at build-time control points
SBOM and component visibility	14	Incomplete adoption, accuracy limits, and dependency graph uncertainty	Use SBOMs as inputs, not as proof of security
Penetration testing adaptation	12	Unclear scope and limited supply chain-specific guidance	Map testing activities to lifecycle stages
Mitigation and governance	19	Need for layered controls, remediation, and continuous monitoring	Link findings to remediation and monitoring
Case-based evidence	8	SolarWinds, Log4j, and malicious package incidents	Use cases to validate lifecycle blind spots

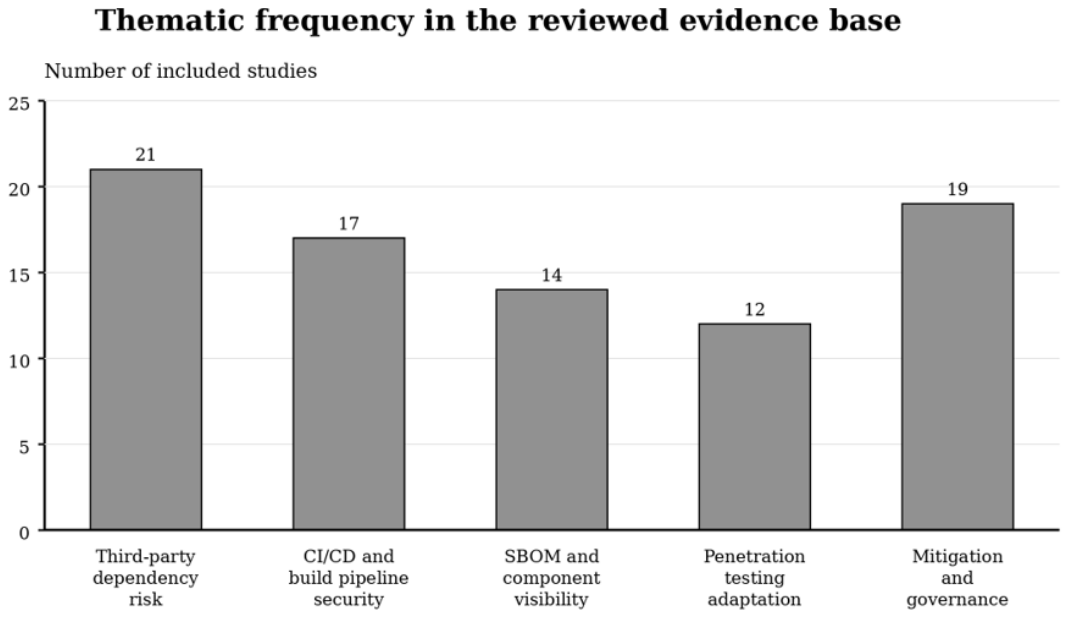

Figure 2. Thematic frequency of reviewed studies

4.1. Finding 1: Existing work emphasises visibility more than exploitability validation

The first finding is that existing software supply chain research places strong emphasis on visibility mechanisms such as SBOMs, dependency graphs, vulnerability databases, and risk assessment frameworks. These mechanisms are necessary because organisations often do not know which direct and transitive components are present in their systems [5], [9], [12], [27]. However, visibility does not automatically establish exploitability.

Several studies show that a listed vulnerable dependency may not be reachable, while a seemingly lower-severity component may create high risk when it appears in a privileged build path or release process [11], [12], [21]. This finding supports the role of penetration testing as a validation layer that converts identified risk into evidence of whether an attack path can actually be exercised.

4.2. Finding 2: Build-stage threats require more operational testing guidance

The second finding is that dependency risks receive more direct attention than build-stage testing guidance. CI/CD pipelines, workflow scripts, build runners, secrets, container images, and artifact generation are repeatedly identified as high-value targets [8], [23]. Yet the literature provides less practical guidance on how penetration testing can be embedded into build-time control points without disrupting delivery.

Practitioners are often more comfortable securing deployment environments because these environments are stable, observable, and easier to monitor. Build environments are different. They are temporary, automated, and difficult to reproduce after execution. This creates a need for lightweight testing patterns such as dependency confusion checks, secret exposure testing, runner permission review, artifact replacement simulation, and provenance validation.

4.3. Finding 3: SBOMs support transparency but do not remove testing needs

The third finding is that SBOMs improve component transparency but cannot independently solve supply chain assurance. Empirical work shows that SBOM adoption, accuracy, maintenance, sharing, and dependency graph completeness remain practical challenges [9], [10], [27]. SBOMs can inform testing scope, but they do not prove that controls resist misuse.

This finding clarifies the relationship between SBOMs and penetration testing. SBOMs help identify what should be examined, while penetration testing validates whether component, build, artifact, or update weaknesses can be exploited. The two mechanisms are complementary rather than interchangeable.

4.4. Finding 4: Supply chain penetration testing faces cost, scale, and coordination limits

The fourth finding is that penetration testing is useful but difficult to scale. Software supply chain risk spans open-source maintainers, third-party vendors, internal developers, security teams, cloud providers, build infrastructure, and release channels. A single test may therefore require coordination across multiple actors [2], [28].

Penetration testing also requires specialised skills and may be costly for small and medium-sized organisations. Continuous manual testing is unlikely to be feasible. A practical approach is to combine automated checks, SBOM monitoring, risk-based prioritisation, periodic expert testing, and evidence-based remediation.

4.5. Finding 5: Case evidence supports lifecycle-based testing implications

The review did not produce new case-based penetration testing results because no empirical testing was conducted. Instead, the case evidence was used analytically to identify where testing would be most relevant across the lifecycle. SolarWinds indicates that trusted build and update paths can fail even when application code appears normal, while Log4j shows how a dependency can become exploitable only when vulnerable functions are reachable in a specific runtime context.

These cases support three testing implications. First, dependency testing should examine reachability and abuse paths rather than only package presence. Second, build and artifact testing should include runner isolation, signing controls, upload permissions, and artifact replacement scenarios. Third, monitoring tests should verify whether alerts, logs, and incident workflows can detect and respond to supply chain compromise. The case evidence therefore strengthens the analytical link between lifecycle exposure and penetration testing practice.

5. Discussion

5.1. Novelty and Value of The Lifecycle-Oriented Perspective

The novelty of this review lies in linking penetration testing to specific software supply chain lifecycle stages. Previous reviews have classified attacks, summarised mitigation strategies, or examined SBOM adoption [4]-[6], [9]. This review extends those discussions by synthesising penetration testing as a dynamic validation practice that can be mapped to dependency selection, development environments, build pipelines, artifact repositories, distribution mechanisms, and monitoring.

This perspective adds value because it helps distinguish between risk discovery and risk validation. A vulnerability scanner or SBOM may identify a potential issue, but penetration testing can determine whether the issue is reachable, exploitable, and insufficiently controlled. This distinction is particularly important in automated development ecosystems where build-time compromise may remain hidden until after release.

5.2. Lifecycle-Oriented Synthesis Model

Figure 3 presents a synthesis model derived from the reviewed themes. It should be interpreted as a conceptual synthesis model, not as a validated operational framework. The model was derived from recurring evidence on dependency risk, CI/CD threats, SBOM limitations, secure development controls, artifact integrity, and monitoring needs.

The model shows that penetration testing can support the software supply chain by translating visibility into validation evidence. The left side represents the evidence base and thematic synthesis. The centre maps testing opportunities across lifecycle stages. The right side shows the expected validation focus and security outcomes. The feedback loop indicates that test results should support remediation and monitoring.

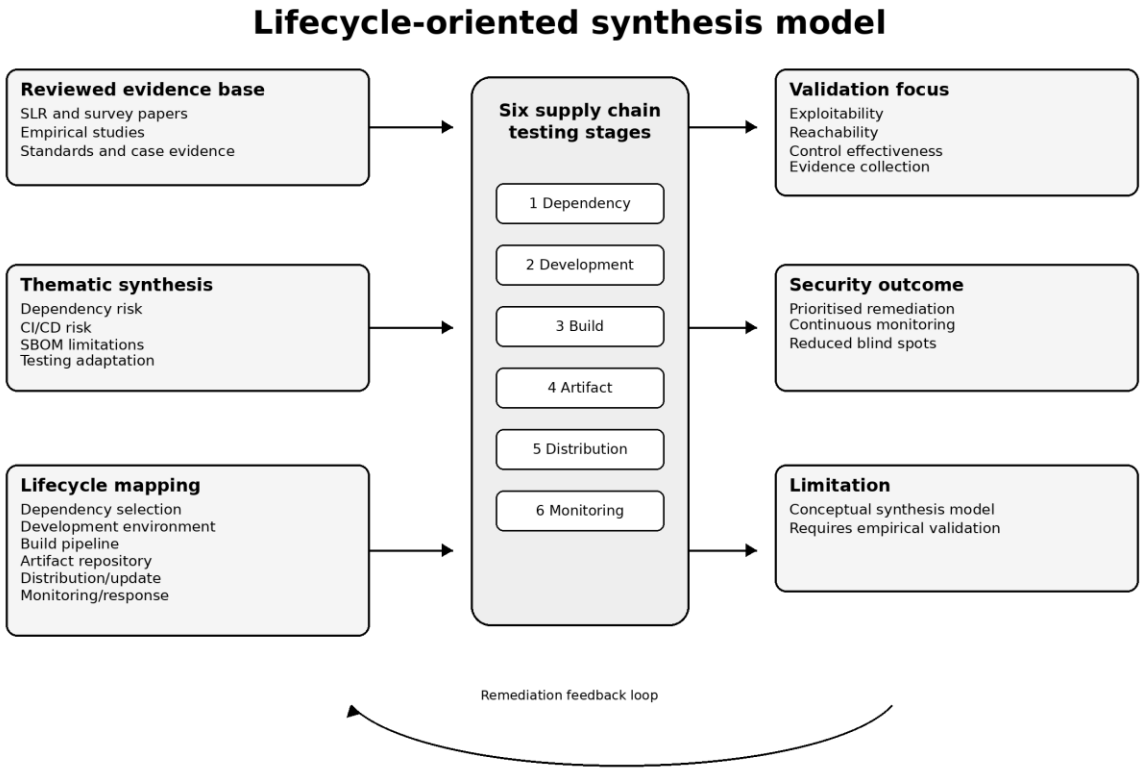

Figure 3. Lifecycle-oriented synthesis model derived from the reviewed literature

The technical interpretation of the model is summarised in Table 6. The testing practices are intentionally framed as validation activities rather than as general controls. For example, dependency confusion testing examines whether package resolution can be abused, repository permission testing checks whether development access boundaries can be bypassed, CI/CD testing examines whether runners and tokens can be misused, and artifact testing verifies whether signed or trusted deliverables can be replaced.

Table 6. Lifecycle mapping of penetration testing activities

Lifecycle stage	Risk tested	Penetration testing focus	Validation evidence
Dependency selection	Malicious or vulnerable package	Package trust, dependency confusion, reachability	Verified package source and exploitability
Development environment	Credential misuse and weak access control	Repository permissions, secrets, branch protection	Confirmed access boundaries
Build pipeline	Build tampering	Runner isolation, workflow scripts, token use	Evidence of build integrity

Lifecycle stage	Risk tested	Penetration testing focus	Validation evidence
Artifact repository	Artifact replacement	Upload rights, signing, provenance	Verified artifact authenticity
Distribution/update	Update hijacking	Signature validation and update integrity	Secure delivery confirmed
Monitoring and response	Delayed detection	Logging, alerts, incident workflow	Response capability confirmed

5.3. SolarWinds and Log4j As Lifecycle-Based Analytical Cases

SolarWinds and Log4j illustrate different weaknesses in the software supply chain lifecycle. SolarWinds demonstrates how a trusted build and update process can be compromised, while Log4j demonstrates how a widely used dependency can create systemic exposure across many organisations [24], [29].

From a lifecycle-oriented penetration testing perspective, SolarWinds highlights the need to test build isolation, artifact integrity, signing controls, and update-channel trust. Log4j highlights the need to test dependency reachability, SBOM accuracy, vulnerable function exposure, and incident response readiness. These cases show that visibility alone is insufficient unless paired with validation and remediation.

Table 7. Lifecycle interpretation of SolarWinds and Log4j

Incident	Lifecycle weakness	What penetration testing could validate	Lesson for the synthesis model
SolarWinds	Build and update-channel compromise	Build isolation, artifact integrity, signing controls, and release-channel trust	Build and release controls require dynamic validation, not only policy assurance
Log4j	Transitive dependency exposure and weak component visibility	Dependency reachability, SBOM accuracy, vulnerable function exposure, and response readiness	Component visibility must be paired with exploitability validation and incident response

5.4. Limitations and Future Research

This review has limitations. The lifecycle model is conceptual and derived from literature synthesis rather than empirical validation in production CI/CD environments. The corpus remains limited to English-language sources and studies that directly address the review scope; therefore, the findings should be interpreted as a structured conceptual synthesis rather than as exhaustive empirical evidence.

Accordingly, the lifecycle model should be used as an organising and analytical tool for future validation work, not as proof that a specific penetration testing programme will produce the same outcomes in every organisation. Its practical value should be tested through controlled CI/CD experiments, organisational case studies, and practitioner evaluation.

Future research should empirically evaluate lifecycle-based penetration testing in controlled CI/CD environments, compare the effectiveness of automated and manual supply chain tests, measure SBOM accuracy across ecosystems, and

develop lightweight testing approaches for small and medium-sized organisations. More evidence is also needed on how organisations translate penetration testing findings into coordinated supply chain remediation.

6. Conclusions

This paper reviewed software supply chain security through the specific lens of penetration testing. The synthesis shows that supply chain risks are distributed across dependencies, repositories, CI/CD pipelines, artifact storage, update mechanisms, and monitoring processes. These risks cannot be fully addressed through static analysis, compliance checks, or SBOM visibility alone.

The main conclusion is that penetration testing is useful as a dynamic validation practice. It helps move software supply chain security from identifying possible risks to testing whether those risks are exploitable and whether controls work as expected. This is especially relevant for dependency confusion, build-pipeline tampering, artifact replacement, weak signing, and update-channel compromise.

At the same time, penetration testing has limitations. It can be difficult to define test boundaries, reproduce short-lived build environments, access third-party components, and maintain continuous coverage. It also requires specialist knowledge and may be resource-intensive for smaller organisations. These limitations mean that penetration testing should complement, rather than replace, SBOMs, vulnerability scanning, secure CI/CD practices, and continuous monitoring.

A lifecycle-oriented approach is therefore recommended as a conceptual synthesis rather than a fully validated operational framework. Future empirical work is needed to test the model in real software engineering environments and to determine how it can be scaled across different organisational contexts.

Author Contributions: Abdiqani Cusman Abdalla: Conceptualization, methodology, formal analysis, investigation, writing-original draft (discussion, conclusion, acknowledgments, funding); **Yuchong Cui:** Conceptualization, formal analysis, investigation, writing-original draft (abstract, introduction, materials and methods, results, author contributions); **Ihab Hussein Al Musawi:** Conceptualization, methodology, formal analysis, investigation, writing-original draft (literature review, conflicts of interest, data availability).

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: The authors would like to thank all members of the School of Computing who contributed to and supported this study. This research was conducted as part of the Hacking and Penetration Testing Project and was supported by Universiti Utara Malaysia.

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

References

[1] M. Souppaya, K. Scarfone, and D. Dodson, *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*, NIST Special Publication (SP) 800-218. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2022. doi: 10.6028/NIST.SP.800-218.

[2] J. Boyens, A. Smith, N. Bartol, K. Winkler, A. Holbrook, and M. Fallon, *Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations*, NIST Special Publication (SP) 800-161 Rev. 1. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2022. doi: 10.6028/NIST.SP.800-161r1.

[3] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, *Technical Guide to Information Security Testing and Assessment*, NIST Special Publication (SP) 800-115. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2008. doi: 10.6028/NIST.SP.800-115.

[4] L. Williams *et al*., "Research directions in software supply chain security," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, Art. no. 146, pp. 146:1-146:38, 2025, doi: 10.1145/3714464.

[5] B. M. Reichert and R. R. Obelheiro, "Software supply chain security: A systematic literature review," *Int. J. Comput. Appl.*, vol. 46, no. 10, pp. 853-867, 2024, doi: 10.1080/1206212X.2024.2390978.

[6] B. Gokkaya, L. Aniello, and B. Halak, "Software supply chain: A taxonomy of attacks, mitigations and risk assessment strategies," *J. Inf. Secur. Appl.*, vol. 97, Art. no. 104324, 2026, doi: 10.1016/j.jisa.2025.104324.

[7] V. Casola, A. De Benedictis, C. Mazzocca, and R. Montanari, "Secure software development and testing: A model-based methodology," *Computer & Security.*, vol. 137, Art. no. 103639, 2024, doi: 10.1016/j.cose.2023.103639.

[8] Z. Pan *et al*., "Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines," *IEEE Trans. Dependable Secure Computing*, vol. 21, no. 1, pp. 403-418, 2024, doi: 10.1109/TDSC.2023.3253572.

[9] S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, "On the adoption of software bill of materials in open-source software projects," *Journal of Systems and Software*, vol. 230, Art. no. 112540, 2025, doi: 10.1016/j.jss.2025.112540.

[10] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, "An empirical study on software bill of materials: Where we stand and the road ahead," in *Proc. IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2630-2642, doi: 10.1109/ICSE48619.2023.00219.

[11] N. Zahan, "Software supply chain risk assessment framework," in *Proc. IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2023, pp. 251-255, doi: 10.1109/ICSE-Companion58688.2023.00068.

[12] A. German Marquez, A. F. Viso, and J. E. Gallardo, "Vulnerability impact analysis in software project dependencies based on satisfiability modulo theories," *Computers & Security*, vol. 139, Art. no. 103669, 2024, doi: 10.1016/j.cose.2023.103669.

[13] D. Bifolco, G. Canfora, A. Di Sorbo, S. Forootani, M. Martinez, and C. A. Visaggio, "An empirical study on the accuracy of GitHub's dependency graph and the nature of its inaccuracy," *Information and Software Technology*, vol. 187, Art. no. 107854, 2025, doi: 10.1016/j.infsof.2025.107854.

[14] B. Pfretzschner and L. ben Othmane, "Identification of dependency-based attacks on Node.js," in *Proc. International Conference on Availability, Reliability and Security (ARES)*, 2017, pp. 1-6, doi: 10.1145/3098954.3098991.

[15] S. Zhang, X. Zhang, X. Ou, L. Chen, N. Edwards, and J. Jin, "Assessing attack surface with component-based package dependency," in *Network and System Security*, Cham: Springer, 2015, pp. 405-417, doi: 10.1007/978-3-319-25645-0_27.

[16] J. Martinez and J. M. Duran, "Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study," *International Journal of Safety and Security Engineering*, vol. 11, no. 5, pp. 537-545, 2021, doi: 10.18280/ijsse.110505.

[17] K.-F. Cheung, M. G. H. Bell, and J. Bhattacharjya, "Cybersecurity in logistics and supply chain management: An overview and future research directions," *Transportation Research Part E: Logistics and Transportation Review*, vol. 146, Art. no. 102217, 2021, doi: 10.1016/j.tre.2020.102217.

[18] A. Adem, E. Cakit, M. Dagdeviren, B. Mrugalska, and W. Karwowski, "Understanding security challenges in the software supply chain through causal relationships," *PLOS ONE*, vol. 21, no. 3, Art. no. e0344098, 2026, doi: 10.1371/journal.pone.0344098.

[19] M. Sroor, R. Mohanani, R. Colomo-Palacios, S. Dasanayake, and T. Mikkonen, "Managing security issues in software containers: From practitioners' perspective," *Journal of Systems and Software*, vol. 231, Art. no. 112616, 2026, doi: 10.1016/j.jss.2025.112616.
[20] M. Egana Aranguren, J. T. Fernandez-Breis, B. Leon Balentzia, M. Rompe, and A. Garcia Castro, "A comprehensive view of software vulnerability risks through enterprise knowledge graphs," *Computers & Security*, vol. 163, Art. no. 104815, 2026, doi: 10.1016/j.cose.2025.104815.
[21] A. Adhikari and P. Kulkarni, "Survey of techniques to detect common weaknesses in program binaries," *Cyber Security and Applications*, vol. 3, Art. no. 100061, 2025, doi: 10.1016/j.csa.2024.100061.
[22] Y. Liu, D. Tiwari, C. Bogdan, and B. Baudry, "Detecting and removing bloated dependencies in CommonJS packages," *Journal of Systems and Software*, vol. 230, Art. no. 112509, 2025, doi: 10.1016/j.jss.2025.112509.
[23] L. Colonna, "The end of open source? Regulating open source under the Cyber Resilience Act and the new Product Liability Directive," *Computer Law & Security Review*, vol. 56, Art. no. 106105, 2025, doi: 10.1016/j.clsr.2024.106105.
[24] 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 *Proc. ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, 2022, pp. 15-24, doi: 10.1145/3560835.3564556.
[25] I. Rahman et al., "S3C2 summit 2024-09: Industry secure software supply chain summit," *arXiv* preprint arXiv:2505.10538, 2025, doi: 10.48550/arXiv.2505.10538.
[26] C. Soto-Valero, T. Durieux, and B. Baudry, "A longitudinal analysis of bloated Java dependencies," in *Proc. 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2021, pp. 1021-1031, doi: 10.1145/3468264.3468589.
[27] A. Javan Jafari, D. E. Costa, E. Shihab, and R. Abdalkareem, "Dependency update strategies and package characteristics," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 149:1-149:29, 2023, doi: 10.1145/3603110.
[28] A. Latsiou and C. Lambrinoudakis, "Cyber supply chain risk management: From threats to treatment," *International Journal of Information Security*, vol. 25, Art. no. 40, 2026, doi: 10.1007/s10207-025-01207-9.
[29] 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*, Cham: Springer, 2020, pp. 23-43, doi: 10.1007/978-3-030-52683-2_2.
[30] P. Ladisa, S. E. Ponta, A. Sabetta, M. Martinez, and O. Barais, "Journey to the center of software supply chain attacks," *IEEE Security & Privacy*, vol. 21, no. 5, pp. 40-48, 2023, doi: 10.1109/MSEC.2023.3302066.
[31] P. Ladisa, S. E. Ponta, N. Ronzoni, M. Martinez, and O. Barais, "On the feasibility of cross-language detection of malicious packages in npm and PyPI," in *Proc. 39th Annual Computer Security Applications Conference (ACSAC)*, 2023, pp. 1-14, doi: 10.1145/3627106.3627138.
[32] P. Ladisa, H. Plate, M. Martinez, and O. Barais, "Risk explorer for software supply chains: Understanding the attack surface of open-source based software development," in *Proc. ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*, 2022, pp. 1-10, doi: 10.1145/3560835.3564546.
[33] A. Decan, T. Mens, A. Zerouali, and C. De Roover, "Back to the past: Analysing backporting practices in package dependency networks," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2953-2971, 2022, doi: 10.1109/TSE.2021.3112204.
[34] T. Mens, "An overview and catalogue of dependency challenges in open source software package distributions," *Science of Computer Programming*, vol. 208, Art. no. 102656, 2021, doi: 10.1016/j.scico.2021.102656.