Defining Software Development Pipeline Code Attacks

Paul Gazzillo
University of Central Florida
Email: paul.gazzillo@ucf.edu

***Abstract*—In this paper, we define software development pipeline code attacks; outline a threat model; and analyze several high-profile attacks.**

1. Software Development Pipelines

The software development life cycle defines software development as a set of distinct phases [1], [2]. A software development pipeline automates the execution of the phases of a software development life cycle. Pipeline automation is often associated with specific development practices, such as Agile [3], DevOps [4], [5], [6], and Continuous Integration, Delivery, and Deployment (CI/CD) [7], [8]. The automation of software development, however, long predates modern development practices. Make was designed for development automation, dates to the 1970s [9], and is still used, particularly in critical systems software [10], [11], [12]. We use the term *software development pipeline*, or just *pipeline* for short, to refer to the whole range of practices for development automation. We use the term *pipeline code* to mean the implementation of a software development pipeline.

There are many special-purpose languages and tools for implementing pipelines [13], [14], [15], [16], [17], [18], [19], [20], [21], [22] and many third-party services that host and execute pipelines [20], [23], [24], [25], [26]. But, strictly speaking, no special languages or platforms are needed to implement and run pipelines. General-purpose languages like bash and python are also used in pipeline code [10]. What distinguishes pipeline code from a software product's program code is not the implementation language but in how the language is used. Pipeline code automates software development practices that produce a software artifact, while program code ultimately ends up as part the resulting software artifact.

To help understand software development pipelines, we invoke the distinction between programs and processes, a fundamental concept in operating systems [27]. In the context of software development pipelines, the pipeline code is a program, i.e., the set of instructions that automate development. The process is then the running pipeline code, which we callt the *pipeline execution*. The computing system that runs the pipeline code is the *pipeline executor*. Figure 1 illustrates the high-level workflow of pipeline code implementation and execution. Developers write pipeline code ①, then send it to a pipeline executor to execute the pipeline ②. The pipeline executes ③ and produces the resulting software artifacts ④.

2. Pipeline Code Attacks

Pipeline code attacks work by first gaining access to modify the pipeline code ① (Figure 1). We call this a *malicious change* to pipeline code. Attackers have and continue to use a wide variety of vectors to make malicious changes to pipeline code, which is the subject of previous taxonomies that include pipeline attacks [28], [29], [30], [31], [32]. Access to pipeline code gives attackers access, albeit indirectly, to the pipeline executor ②. Third-party platforms providing CI/CD services protect themselves from outside intrusion and malicious pipelines through virtualization and network security [33], [34]. But within that protected boundary, the running pipeline code is free to access whatever resources the developer provides it, including source code, software artifacts, and credentials.

Attackers, therefore, have no need for direct access to the pipeline executor or sensitive pipeline resources. Instead they modify the pipeline code to indirectly gain access to these resources ③, in spite of the protections the platform provides around the pipeline executor. Attackers can also alter the resulting software artifacts produced by the pipeline ④, incorporating backdoors or otherwise weakening their security [35], [36], [37], [38].

Pipeline code, like any program, can be understood as performing information flow [39], [40], [41], [42]. When these flows enable an attacker to gain indirect access to sensitive resources during pipeline execution, we call these *malicious flows*. A *pipeline code attack* is when malicious changes to pipeline code cause malicious flows in the pipeline executor. Our understanding categorizes attacks by these malicious flows and examines the scope of attacks to which this definition of pipeline code attacks applies.

3. Threat Model

In our threat model, we assume that (1) attackers can access pipeline code, but (2) they cannot directly access the pipeline executor.

(1) Insecure pipeline code. We assume the attacker can modify the pipeline code via *any* vector, known or unknown. Therefore, attackers can make or already have made malicious changes to pipeline code. We do not include direct access to modify software program code, since it is not pipeline code and constitutes a different class of attacks. Moreover, direct access to program code obviates the need for indirect access to the pipeline executor via pipeline code.

© 2026 Paul Gazzillo. All Rights Reserved.

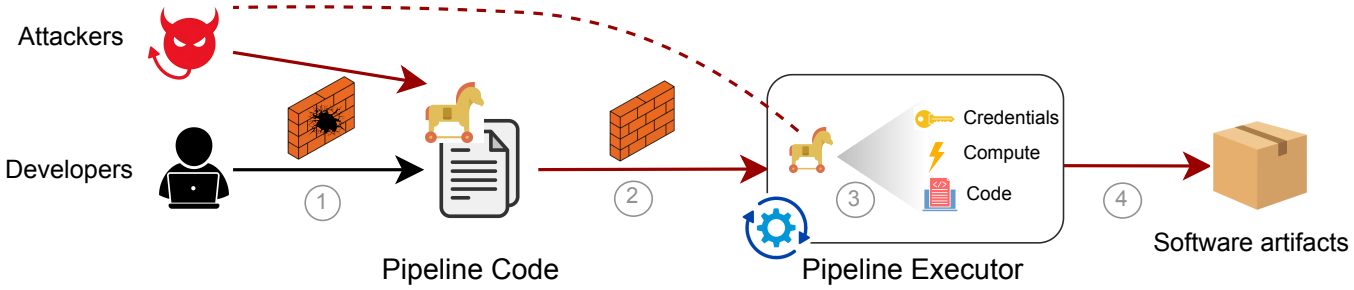

Figure 1. Pipeline code attacks.

(2) Secure pipeline executor. The attacker does not have direct access to the pipeline executor itself, although access to the pipeline code gives them indirect access to the pipeline executor. Direct access to the pipeline executor would also obviate the need to modify pipeline code for indirect access.

This threat model is beneficial, because it does not rely solely on protecting against the myriad vectors attackers use to alter pipeline code, which attackers continue to find and exploit. It aligns with the prevalent practice of using third-party, virtualized pipeline executors [33], [43] that secure the machine executing the pipeline but not the behavior of the code. It reduces the attack surface to only the run-time behavior of the pipeline code during execution, i.e., the malicious flows. Protecting the run-time pipeline behavior adds an additional layer of defense on top of existing defenses against vectors and pipeline executors. Researchers are actively investigating on techniques to defend against malicious pipeline flows via isolation enforcement [42], [44], [45], taint analysis [46], [47], and reproducible builds [48], [49]. Our framing helps researchers understand the scope of defenses against malicious pipeline flows and how to categorize which techniques apply to which categories of attack.

Our threat model differs from those of taxonomies that focus on the vectors that attackers use to make malicious pipeline code changes [28], [29], [30], [31], [32]. In contrast, our threat model assumes these vectors have already been exploited. It also differs from, but overlaps, the traditional software supply chain threat models [32]. This model assumes that the attacker cannot directly change the target software's pipeline code, e.g., via insider threats, but that the software's dependencies can be modified by attacker. In contrast, our threat model assumes attackers can directly access pipeline code. But changes to software program code are out of scope, since it is not pipeline code. However, supply chain attacks where the pipeline imports and executes a software dependency's pipeline code, e.g., dependency confusion [50], are in scope, since that allows an attacker to make malicious changes to pipeline code.

4. Real-World Examples

Figure 2 illustrates a typical software development pipeline [1], [24], [51]. The overall pipeline automator coordinates the phases of the development pipeline, including build, test, and deploy. Information flows into, out of, and between the phases. The black arrows between phases in Figure 2 represent legitimate flows among development phases. For instance, the build phase reads source code, executes a compiler, and writes program binaries; the test phase reads program binaries and the test files, then runs them; and the deployment phase reads credentials and uploads the packaged software to an external server.

Pipeline code attacks, however, exploit the combination of access to sensitive information and freedom to execute powerful tools, like shells and scripting languages, to cause malicious flows during pipeline execution. The red arrows in Figure 2 represent malicious flows that are caused by seven high-profile pipeline code attacks. The target platforms and the vectors used in these attacks vary widely. But by examining the behavior of the pipeline caused by an attacker's pipeline code changes, a common theme emerges. They all cause the pipeline to make unexpected flows of information that have malicious effects, including credential theft, software backdoors, and resource theft.

Take the four credential theft attacks, HackerBot-Claw, CodeCov, CIDER, and xssfox's. In all four cases, the attack modifies the pipeline to call `curl` or similar networking commands to send credentials to an attacker-controlled server during pipeline execution. Classifying by vector, however, separates these attacks into distinct categories that each entail different defense strategies. CIDER and HackerBot-Claw both use the Direct Pipeline Poisoning Execution (D-PPE) [52] vector. CIDER was an early demonstration of the attack by a researcher [53], while HackerBot-Claw was a malicious autonomous AI agent that stole credentials from numerous high-profile repositories, including those for Trivy, Microsoft, DataDog, and others [54].

D-PPE attacks work by first forking the target repository then modifying the forked copy of the CI/CD pipeline configuration file. Next the attacker makes a pull request that triggers execution of the modified CI/CD pipeline of the target repository. The pull request securely passes developer credentials to CI/CD pipeline executor, which is hardened against intrustion, to access the source repository. The malicious pipeline code instructs to pipeline executor to read and send these credentials to the attacker during execution. One defense strategy for D-PPE is to review changes to pipeline configuration file before allowing execution [52]. But xssfox

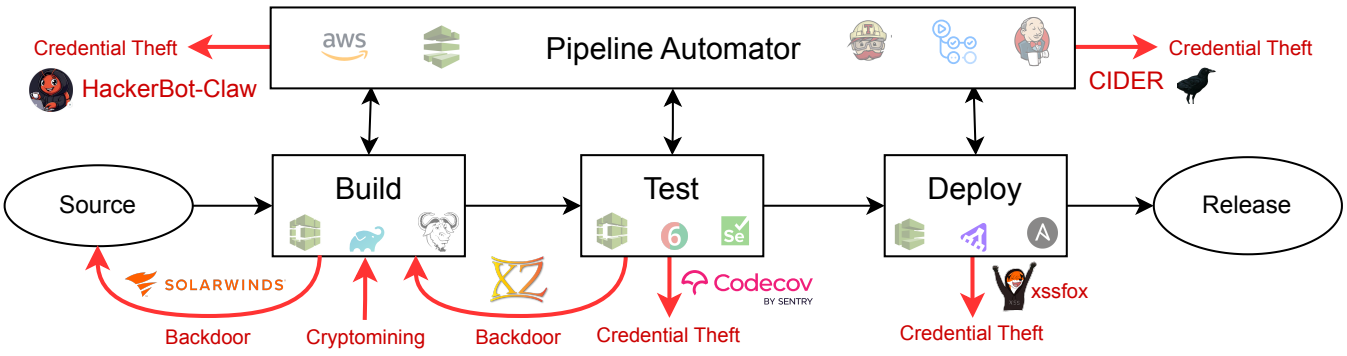

Figure 2. Several examples of pipeline code attacks and their malicious flows.

introduced a variation, called Indirect PPE (I-PPE) [52], [55] that modifies a deployment script called by the pipeline, circumventing reviews of the pipeline configuration file. Figure 2 illustrates this distinction by sourcing xssfox's credential theft flow from the deploy phase instead.

For both types of PPE, defense recommendations include limiting who is permitted to trigger the CI/CD pipeline. But this strategy only applies to public CI/CD pipelines for open-source software. In contrast, the CodeCov attack exfiltrated developer without making pull requests. CodeCov is a popular tool for measuring code coverage during testing that developers configured their pipelines to automatically download and run during the testing phase. [56]. Instead of targeting individual developers, attackers targeted the CodeCov tool itself. Attackers extracted credentials to the site hosting the CodeCov script from a public docker image and modified it to include a call to `curl` [57] to send credentials to an attacker-controlled server. As a consequence, thousands of credentials were exfiltrated, and hundreds of customer sites were breached [58].

Defenses for PPE attacks are orthogonal to those for CodeCov's vector, which involve traditional credential management and validitation recommendations [59], [60], [61], [62] rather than CI/CD access permissions. But the behavior of the malicious pipeline code shares similarities that crosscut the different vectors. They all cause an unpermitted flows of credentials from the pipeline executor to an external server. Enforcing flow permissions would prevent the negative effects of attacks [42], [45], [46], even if an attack can successfully exploit a vector to modify the pipeline code. Understanding attacks as malicious flows through the pipeline executor can help identify common defenses that apply to all pipeline executions, independent of the specifics of the pipeline's implementation or platform.

Explaining attacks by their malicious flows also broadens the scope of pipeline code attacks. Attacks that appear distinct, because they involve vectors or different outcomes, become variations of the same problem: the pipeline code causes flows that violate access permissions during execution. For instance, the XZ Utils attack resulted in an SSH server backdoor in the testing release of Debian [63], one of the most widely-used Linux distributions. The XZ Utils attack was deployed via malicious changes to its pipeline code. But XZ Utils pipeline does not use a CI/CD service, making traditional CI/CD vectors inapplicable. The attackers instead made publicly-visible commits of malicious backdoor code hidden as compression test files [63], [64], [65]. The attackers then used social engineering tactics to make changes to the build phase the pipeline to unpack and link the malicious code into the XZ Utils library [66], [67]. Those building from source, e.g., the Debian maintainers, unwittingly ran the malicious pipeline, resulting in an XZ Utils library binary that contained the SSH backdoor.

At first glance, XZ Utils appear very different from the previously-described CI/CD attacks: XZ Utils does not use a CI/CD service, and the attackers used different vectors. While XZ Utils has no official CI/CD pipeline, it still has a pipeline, albeit implemented with autotools and Makefiles and run by whomever wants to build the software from source. But XZ Utils's similarity to CI/CD pipeline attacks can be seen in the malicious flows caused by the attack. The malicious pipeline code causes the build phase to read test files while compiling and linking source code, shown by the red arrow from test phase to build phase in Figure 2, atypical behavior for a build phase. Recent work demonstrates that restricting the build phase's permissions when running the malicious XZ Utils pipeline prevents the attacks [42].

Framing pipeline code attacks as malicious flows during pipeline execution reveals their similaries and helps identify common defenses, such as enforcing access permissions. Like XZ Utils, the SolarWinds SUNBURST attack leveraged the development pipeline to insert a backdoor, albeit using a completely different vector from XZ Utils to change the pipeline. One of the largest attacks ever at the time, hundreds of government agencies and corporations were affected [36], [68], [69]. Attackers gained access to the build system of the Orion software, likely through weak passwords [70]. The build system was modified to insert backdoor code into the source code [71], [72], yet another kind of malicious flow during pipeline execution.

References

[1] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.

[2] IBM, "What is the Software Development Lifecycle (SDLC)?" https://www.ibm.com/think/topics/sdlc, accessed: 2026-05-05.

[3] A. Alliance, "What is Agile?" https://agilealliance.org/agile101/, 2026, accessed: 2026-02-03.

[4] IBM, "What is DevOps?" https://www.ibm.com/think/topics/devops, 2026, accessed: 2024-02-05.

[5] A. W. Service, "What is DevOps?" https://aws.amazon.com/devops/what-is-devops/, 2026, accessed: 2026-02-03.

[6] "What is devops?" https://learn.microsoft.com/en-us/devops/what-is-devops.

[7] IBM, "What Are CI/CD And The CI/CD Pipeline?" https://www.ibm.com/think/topics/ci-cd-pipeline, accessed: 2026-05-05.

[8] R. Hat, "What is CI/CD?" https://www.redhat.com/en/topics/devops/what-is-ci-cd, 2025, accessed: 2025-01-22.

[9] S. I. Feldman, "Make — a program for maintaining computer programs," *Software: Practice and Experience*, vol. 9, no. 4, pp. 255–265, 1979. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380090402

[10] "The linux kernel," https://www.kernel.org.

[11] "Gnu coreutils," https://www.gnu.org/software/coreutils/coreutils.html.

[12] "Gnu make," https://www.gnu.org/software/make/manual/make.html, 2023.

[13] "Travis ci build config reference," https://config.travis-ci.com/.

[14] "Build specification reference for codebuild - aws codebuild," https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html.

[15] "Yaml schema reference for azure pipelines," https://learn.microsoft.com/en-us/azure/devops/pipelines/yaml-schema/?view=azure-pipelines.

[16] "Overview - configuration language — terraform — hashicorp developer," https://developer.hashicorp.com/terraform/language.

[17] "Pipeline syntax," https://www.jenkins.io/doc/book/pipeline/syntax/.

[18] "Gnu autoconf," https://www.gnu.org/software/autoconf/.

[19] "Github actions," https://docs.github.com/en/actions.

[20] "Buildbot," https://www.buildbot.net/.

[21] G. Make, "https://www.gnu.org/software/make/."

[22] "JUnit," junit.org/.

[23] "About continuous integration with github actions," https://docs.github.com/en/actions/about-github-actions/about-continuous-integration-with-github-actions.

[24] "What is ci? - continuous integration explained - aws," https://aws.amazon.com/devops/continuous-integration/.

[25] "Use continuous integration," https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration.

[26] "Jenkins," https://www.jenkins.io/.

[27] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 1st ed. Arpaci-Dusseau Books, November 2023.

[28] "Owasp top 10 ci/cd security risks," https://owasp.org/www-project-top-10-ci-cd-security-risks/.

[29] R. Zmuda, R. Graves, M. Shepherd, and S. Brookes, "Sok: Understanding ci/cd security: A comprehensive review of architecture, attacks, and defenses," in *2025 IEEE Secure Development Conference (SecDev)*, 2025, pp. 58–68.

[30] B. Gokkaya, L. Aniello, and B. Halak, "Software supply chain: A taxonomy of attacks, mitigations and risk assessment strategies," *Journal of Information Security and Applications*, vol. 97, p. 104324, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212625003606

[31] 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*, C. Maurice, L. Bilge, G. Stringhini, and N. Neves, Eds. Cham: Springer International Publishing, 2020, pp. 23–43.

[32] 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)*, 2023, pp. 1509–1526.

[33] "Security in aws codepipeline - aws codepipeline," https://docs.aws.amazon.com/codepipeline/latest/userguide/security.html.

[34] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the security of github CI workflows," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2747–2763. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/koishybayev

[35] "Xz utils," https://github.com/tukaani-project/xz.

[36] U. G. A. Office, "Solarwinds cyberattack demands significant federal and private-sector response (infographic)," https://www.gao.gov/blog/solarwinds-cyberattack-demands-significant-federal-and-private-sector-response-infographic, 2021.

[37] "Xz: Can you spot the single character that disabled linux landlock?" https://news.ycombinator.com/item?id=39874404.

[38] "CVE-2023-43608," https://www.cve.org/CVERecord?id=CVE-2023-43608.

[39] T. Terauchi and A. Aiken, "Secure Information Flow as a Safety Problem," in *Proceedings of the 12th International Static Analysis Symposium*, ser. Lecture Notes in Computer Science, vol. 3672. Springer International Publishing, 2005, pp. 352–367.

[40] A. Sabelfeld and A. Myers, "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5–19, 2003.

[41] D. E. Denning and P. J. Denning, "Certification of programs for secure information flow," *Commun. ACM*, vol. 20, no. 7, p. 504–513, Jul. 1977. [Online]. Available: https://doi.org/10.1145/359636.359712

[42] B. Pappas and P. Gazzillo, "Build code is still code: Finding the antidote for pipeline poisoning," in *Proceedings of the 48th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '26, 2026. [Online]. Available: https://doi.org/10.1145/3786582.3786799

[43] "Github-hosted runners," https://docs.github.com/en/actions/concepts/runners/github-hosted-runners.

[44] Z. Pan, W. Shen, X. Wang, Y. Yang, R. Chang, Y. Liu, C. Liu, Y. Liu, and K. Ren, "Ambush from all sides: Understanding security threats in open-source software ci/cd pipelines," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 1, pp. 403–418, 2024.

[45] S. Hamer, J. Bowen, M. N. Haque, R. Hines, C. Madden, and L. Williams, "Closing the chain: How to reduce your risk of being solarwinds, log4j, or xz utils," 2025. [Online]. Available: https://arxiv.org/abs/2503.12192

[46] S. Muralee, I. Koishybayev, A. Nahapetyan, G. Tystahl, B. Reaves, A. Bianchi, W. Enck, A. Kapravelos, and A. Machiry, "Argus: a framework for staged static taint analysis of github workflows and actions," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USA: USENIX Association, 2023.

[47] G. Tystahl, J. Ghebremichael, S. Muralee, S. Cherupattamoolayil, A. Bianchi, A. Machiry, A. Kapravelos, and W. Enck, " Cosseter: GitHub Actions Permission Reduction Using Demand-Driven Static Analysis ," in *2026 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2026, pp. 1243–1260. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00067

[54] C. Hamsen, K. Serrania, and C. Tafani-Dereeper, "When an ai agent came knocking: Catching malicious contributions in datadog's open source repos," https://www.datadoghq.com/blog/engineering/stopping-hackerbot-claw-with-bewaire/, 2026.

[55] "Build pipeline security," https://sprocketfox.io/xssfox/2021/02/18/pipeline/.

[56] "Uploading reports to codecov using the codecov cli," [Deprecating] BashUploader.

[57] "Codecov breach: All questions answered," https://www.sisainfosec.com/blogs/codecov-breach-all-questions-answered/.

[58] "Codecov hackers breached hundreds of restricted customer sites - sources," https://www.reuters.com/technology/codecov-hackers-breached-hundreds-restricted-customer-sites-sources-2021-04-19/.

[59] "Cicd-sec-5: Insufficient pbac (pipeline-based access controls) — owaspp foundation," https://owaspp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-05-Insufficient-PBAC.

[60] "Cicd-sec-6: Insufficient credential hygiene — owaspp foundation," https://owaspp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-06-Insufficient-Credential-Hygiene.

[61] "Cicd-sec-8: Ungoverned usage of 3rd party services — owaspp foundation," https://owaspp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-08-Ungoverned-Usage-of-3rd-Party-Services.

[62] "Cicd-sec-9: Improper artifact integrity validation," https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-09-Improper-Artifact-Integrity-Validation.

[63] "oss-security - backdoor in upstream xz/liblzma leading to ssh server compromise," https://www.openwall.com/lists/oss-security/2024/03/29/4.

[64] "git.tukaani.org - xz.git/commitdiff," https://git.tukaani.org/?p=xz.git;a=commitdiff;h=cf44e4b7f5dfdbf8c78aef377c10f71e274f63c0.

[65] "git.tukaani.org - xz.git/commitdiff," https://git.tukaani.org/?p=xz.git;a=commitdiff;h=74b138d2a6529f2c07729d7c77b1725a8e8b16f1.

[66] "The xz attack shell script," https://research.swtch.com/xz-script.

[67] D. Goodin, "What we know about the xz utils backdoor that almost infected the world," https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/.

[68] Reuters, "Solarwinds hack was 'largest and most sophisticated attack' ever: Microsoft president," https://www.reuters.com/article/technology/solarwinds-hack-was-largest-and-most-sophisticated-attack-ever-microsoft-pres-idUSKBN2AF03Q/.

[69] Bloomberg, "At least 200 victims identified in suspected russian hacking," https://www.bloomberg.com/news/articles/2020-12-19/at-least-200-victims-identified-in-suspected-russian-hacking.

[48] 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 *Proceedings of the 28th USENIX Conference on Security Symposium*, ser. SEC'19. USA: USENIX Association, 2019, p. 1393–1410.

[49] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022.

[50] A. Birsan, "Dependency confusion: How i hacked into apple, microsoft and dozens of other companies," https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610.

[51] "Standard Targets for Users," https://www.gnu.org/software/make/manual/html_node/Standard-Targets.html.

[52] "Cicd-sec-4: Poisoned pipeline execution (ppe) — owasp foundation," https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-04-Poisoned-Pipeline-Execution.

[53] T. Welton, "Def con 25 - spaceb0x - exploiting continuous integration (ci) and automated build systems," https://www.youtube.com/watch?v=mpUDqo7tIk8.

[70] "SolarWinds hack explained: Weak password "solarwinds123" cause of SolarWinds Hack," https://specopssoft.com/blog/solarwinds-hack-weak-password-solarwinds123-cause/.

[71] CISA, "Solarwinds compromise," https://www.cisa.gov/eviction-strategies-tool/dialog-content/TMPL0002, accessed: 2025-11-06.

[72] SOLARWINDS CORPORATION, "FORM 8-K: CURRENT REPORT," https://www.sec.gov/Archives/edgar/data/1739942/000162828020017451/swi-20201214.htm.