Detecting Unsafe Updates in Software Ecosystems

Yao-Wen Chang
Student ID: 1346258

Supervisor: Christoph Treude
Industry Mentor: Behnaz Hassanshahi (Oracle Labs, Australia)

Word Count: X words

Submission Date: September 17, 2023

Contents

Abstract

This literature review will introduce Continuous Integration/Continuous Delivery (CI/CD). Also, some attack surface and how the malicious attacker exploit these attack surface will be covered in this review. From the developers' and maintainers' point of view, the methods and frameworks are going to be introduced to counter the attack. In this literature review, the framework will focus on Supply Chain Level Security Artifacts (SLSA) which is adopted by Google.

1 Introduction

Continuous IntegrationContinuous Delivery (CI/CD) is a development process for quickly building and testing code changes that helps organizations maintain a consistent code base for their applications while dynamically integrating code changes. Therefore, CI/CD environments are attractive targets for malicious cyber actors (MCAs) whose goals are to compromise information by introducing malicious code into CI/CD applications, gaining access to intellectual property/trade secrets through code theft, or causing denial of service effects against applications.

Recent incidence like the infection of SolarWind's Orien platform [2, 7] which is used to monitor and manage the network is downloaded by thousands customers, including U.S. government agencies, critical infrastrure providers, and private companies.

Section 2 will briefly introduce CI/CD. Section 3 would target the attack surface within the process of CI/CD and the counter method. In section 4, our literature review would introduce SLSA framework from Google, and explain how SLSA can patch the vulnerable CI/CD process. In section 6, the aim and objects of the research will be explained. And the research plan will be introduced in section 7.

2 Definition of CI/CD

CI/CD is a development process for automatically building and testing code changes that support orgamizations maintain a consistent code base. CI involves developers frequently merging code changes into a central repository where automated builds and tests run. Build is the process of converting source code into executable code. Then, running the automated tests against the build. These process will avoid integration challenges that can happen when waiting for release day to merge changes into the release branch. CD defined the process of the releases happened automatically. [6].

The convenience and capabilities of the third-party source code usually brings cybersecurity risks [4]. Software supply chain attacks aim at injecting code into software components to compromise downstream users [2]. Some vulnerabilities will be introduced in the next section.

3 Risk and Defense Method

3.1 Source Code Repository

Obtain Git Repository credentials by dumping the environment Variables. Then, user the stolen secrets to access the Repository and modify the CI/CD configuration. Finally, the code will successfully be injected into the code base, and affects downstream users.

3.2 Build/Test

3.3 Deploy

Java Virtual Machine (JVM) executes Java bytecode and provides strong safety guarantees. However, the unsafe API, "sun.misc.Unsafe", will cause serious security issue if it is misused by the developers. The research [4] studied a large repository, Maven, and analyzed the compiled Java code. The security issues include violating type safety, crashing the virtual machine (VM) , uninitialized objects and so on. These misuse might impact third-party package management service.

These goals are seek to establish the trust in the software supply chain by verifying in- formation about the participants or processes [5]. The complex CI/CD has the potential to catch the attackers more quickly via peer review, because the developers and distributors get a chance to review the code, increasing the chances of malicious code being discovered [3]. Providing cryptographic hashes if the software packages is of significance to verify the software's integrity [3]. Signing the releases of the packages by the providers with the public-key enable the users to verify the

3.4 OWASP Top 10 CI/CD Risks[1]

(1) **Insufficient Flow Control Mechanisms**

Definition: The attacker obtain permissions to a system within CI/CD process. The system did not provide sufficient enforcement for approval and reviewed machnisms.

Impact:

- The attackers can push code to a repository branch, which is automatically deployed to production and manually trigger by the attackers.
- Upload an artifact to the artifact repository, such as a package or container , in the guise of a legitimate artifact created by the build environment and picked up by a deploy pipeline and deployed to production.

Remediation:

- Configure strict branch protection rules
- Limit the usage of auto-merge rules
- Prevent drifts and inconsistencies between the running code in production and its CI/CD origin.

(2) **Inadequate identity and Access Management**

Definition: This risks stem from the difficulties in managing the vast amount of identities. The identies are indentified through personal access token, e-mail, password and so on.

- Overly permissive identities
- Stale identies - Some identies that are not active or no longer require access but have no had their account deactivated.
- External indentities - (1) Employees registered with email from a domain not managed by the organization (2) External collaborators are outside of the organization's control.

Impact:

- Overly permissive accounts leads to a state where the attacker can compromise any user account on any system within the CI/CD pipeline.

Remediation:

- Continuously analyzed and mapped the identies' account to their permissions, and removed the the permissions not necessary to the ongoing work.
- Ensure the identies are aligned to the principle of least privilege, and pre-defined a expiry date for the identies' permissions.
- Prevent the emplyees from using personal email addresses.
- Avoid the shared accounts. Created the dedicated accounts for each specific context.

(3) **Dependency Chain Abuse**z

Definition: Dependency chain abuse refer to an attacker's ability to abuse flaws relating to how engineering workstations and build environments fetch code dependencies. The build system download the malicious package instead of the one intended to be pull. There are four scenarios where the developers might be tricked.

- Dependency confusion - Publication of malicious packages in public repositories with the same names as those private one.
- Obtain the control of the account of the package maintainer in order to upload the malicious version.
- Typosquatting - Publication of a similar names to those popular packages.
- Brandjacking - The malicious packages were consistent with the naming convention with the trusted brand.

Impact: Once the malicious code is running, it can be leveraged for credentials theft and move horizontally through a system and network.

Remediation:

- Ensure the packages are not directly pulled through the internet, but through an internal proxy. And disallow pulling directly from external repositories.
- Verify checksum and signature of the pulled packages.
- Lock the packages' version instead of pulling the latest version.
- Installation scripts should not access to sentitive resources in the build process.
- Always ensure projects contain configuration files of package managers.
- The most important is deploy a quick detection, monitoring and mitigation to avoid further compromise.

(4) **Poisoned Pipeline Execution (PPE)**

Definition: The attacker access to the source control systems, but without access to the build environment, is able to manipulate the build process by injecting malicious code into the build configuration file. There are three type of PPE, direct PPE (D-PPE), indirect PPE (I-PPE) and public-PPE (3PE).

In the D-PPE scenario, the attackers modify the CI config files either by submitting a PR or directly pushing to the unprotected remote branch. Since the CI pipeline execution is triggered by push or PR events, and the CI execution is defined by CI Configuration file, the malicious commands run in the build node.

In the I-PPE scenario, the pipeline is configured to pull the CI configuration file from a protected branch or CI build is defined by the CI system instead of the in the file stored in the source code. In those cases, the attackers can still injecting malicious code into the files referenced by the pipeline configuration file.

In most cases, the permissions of the access to the repository are given to the orgamization members. However, in the 3PE scenario, the public repositories are allowed the anonymous to contribute. If the CI pipline runs unreviewed code, the repository is susceptible to the 3PE.

Impact:

- Access to the secret available to the CI job.
- Able to ship code and artifacts futher down the pipeline, in the guise of legitimate code build by the build process.

Remediation:

- Ensure that pipelines running unreviewed code are executed on isolated nodes to prevent exposure of sensitive information.
- To prevent the manipulation of the CI configuration file.
- Remove permissions from the users that do not need them.

(5) **Insufficient PBAC (Pipeline-Based Access Controls)**

Definition: Adversary is able to execute malicious code within the context of the pipeline. The pipeline execution nodes have access to the resources or systems within and outside the execution environment.

Impact: Malicious code is able to run in the context of the pipeline. Probabily, this attack would lead to the exposure of the secret and confidential data.

Remediation:

- Do not use shared node for pipeline with different levels of sensitivity.
- Where applicable, run pipeline jobs on a separate, dedicated node.

(6) **Insufficient Credential Hygiene**

Definition: CI/CD environments are built of multiple systems communicating and authenticating against each other through verifying the credentials. Insufficient credential Hygiene generally means the overly permissive or the credentials are accidently existed in CI/CD pipelines and the code repositories. Some cases are due to the unrotated credentials issue.

Impact: The adversary obtains the credentials to deploy the malicious code and artifacts.
Remediation:

- Conform to the least privilege rule abd granted the exact set of permission.
- Avoid sharing the same sets of credentials across multiple contexts.
- Using temporary credentials. If using static credentials, better periodically roate all the static credentials and detect stale credentials.
- Scoping the credentials to specific source IP to ensure the credentials cannot be used outside the environment even if it is compromised.
- Detect secrets pushed to the code repositories.

(7) **Insecure System Configuration**

Definition: Flaws in the security settings and configuration, which often results in easily compromised by The attackers. For example, the overly permissive network access controls allow the attackers to interact with different domain.

Impact: The flaws may be abused by the attacker to manipulate CI/CD flows, and obtain the sensitive tokens.

Remediation:

- Ensure the network access the systems is aligned with the principle of least access.
- Periodically review all system configuration.
- Ensure permission to pipeline execution nodes follow the least privilege principle.

(8) **Ungoverned Usage of 3rd Party Services**

Definition: 3rd party services are granted access to the organization's assets, such as the CI/CD systems.

Impact: Lack of governance and visibility around 3rd part might allow the write permission on the repository. Then, the flaw is leveraged by the adversary to push the code to the repository.
Remediation:

- Define the scoped context that the 3rd parties are able to access, and with strict ingress and egress filter.
- Established vetting procedures to verify the trustworthiness of the 3rd paries. Prior to being granted access to the environment, the approval of being granted to resources should be verified.

(9) **Improper Artifact Integrity Validation**

Definition: This flaw allow an attacker with access to the CI/CD process to push malicious code or artifacts down the pipeline.

Impact: Execution of malicious code.

Remediation:

- Validate the integrity of resources all the way from development to production.
- Code signubg to prevent unsigned commits from flowing down the pipeline.
- Prior to fetching and using 3rd parties, the hash of the 3rd paries should be culculated and cross referenced against the official published hash of the resource provider.

(10) **Insufficient logging and visibility**

Definition: The risk allow the adversary to carry out malicious activities without being detected. For example, the permission modificaion and execution of builds.

Impact: Fail to detect a breach may face difficulties in mitigation. Time and data are the most valuable commodities to an organization under the attack.

Remediation:

- First, be familiar with different systems involved in the potential threats.
- Make sure all relevant logs are enables.s
- Shipping logs to a centralized location and creating the alerts to detect malicious activities.

3.5 U.S. Department of Defense Recommend [6]

1. **Zero Trust Approach**

   No user, endpoint device or process is fully trusted.

2. **Strong Cryptographic Algorithm**

   Avoid using outdated crytographic algorithm which poses a threat to CI/CD pipelines. The threat includes sensitive data exposure and keys generated across the CI/CD pipeline.

3. **Minimize the Use of Long-Term Credentials**

4. **Add Signature to CI/CD Configuration and Cerify It**

   Ensure the code change is continuously signed, and the signaature is verified throughout CI/CD process. If the signing identity itself is compromised, it undermines trust.

5. **Two-Person Rules for all Code Updates**

   The developer checks in the code which should be reviewed and approved by another developer.

Some of the projects aims at providing single solution that conflates multiple objectives [5].

Despite the previously introduced methods seems to address all the security issue existed in the code base and within the CI/CD, some of them may overemphasize one particular approach to address software supply chain security. without considering compounding factors that impact risk [5].

4 Supply Chain Level Security Artifacts

4.1 What is Software Supply Chain?

Software supply chain is composed multiple components, first-party or third- party libraries, and processes used to develop, build, test, and publish a software artifact [6].

4.2 What is SLSA?

A security framework provides checklist to ensure the integrity of the supply chain. Also, the SLSA level give consumers confidence that the software has not been tamperated with and can be trace back to the source.

4.3 Provenance

SLSA provenance clearly provides the transparent information about the artifacts or the packages. Information such as, who builds this artifact and how the artifact was built from the source. These information will be verified by the package registory or even the customers. The provenance is an attestion in SLSA.

4.4 SLSA Provenance V.S SBOM (Software Bill of Materials)

Provenance and SBOM are somehow similar, so they are easily confused. Provenance is used to assess the trustworthiness and security of the processes used to build and deliver the software artifacts 1. By contrast, SBOM focus on listing software components and their versions 2.

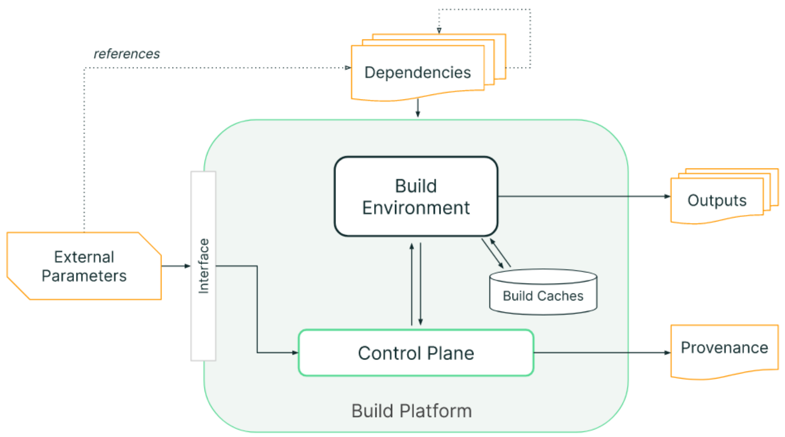
Listing 1: SLSA Provenance

```
[Software Build Provenance]
Build Date: 2023-09-01
Build Environment: Secure, Isolated Environment
Signing Authority: Trusted Certificate Authority (CA)
Signature Verification: Passed
[Supply Chain Processes]
Code Review: Multi-stage code review by security experts
Dependency Scanning: Automated scanning for known vulnerabilities
Build Automation: Continuous Integration/Continuous Deployment (CI/CD) pipeline
Deployment: Automated deployment to secure servers
[Organizations Involved]
Development Team: Responsible for code development
Security Team: Responsible for security reviews and scanning
Operations Team: Responsible for deployment
```

Listing 2: SBOM

```
[MyApp (v1.0)]
Frontend Framework (v2.3)
Database Connector (v1.1)
Authentication Library (v3.0)
Logging Utility (v1.2)
```

4.5 Build Model



4.6 SLSA Security Level

Currently SLSA have defined 4 levels (LV.0 - LV.3), which will be briefly described in Table 1, and working on level 4.

Table 1: SLSA Security Levels

Level	Requirements	Focus
Build L0	None	No security practices are in place.
Build L1	With provenance	Basic security practices are followed, such as code review and basic dependency scanning.
Build L2	Signed provenance, generated by a hosted build platform	More comprehensive security practices are implemented, including in-depth code review, vulnerability scanning, and build verification.
Build L3	Hardened build platform	The highest level of security is maintained, with strict adherence to security practices, automated testing, and supply chain integrity checks.

5 Conclusion

6 Research Aims and Objectives

6.1 Research Aims

This reseach is aim to contribute the Macaron framework, then examine Git repositories with this framework. The statistical conclusion from the examination will be futher discussed. Also, the results will provide the developers and maintainers of the Git repositories with a clear understanding of the vulnerabilities of their CI/CD pipelines configuration. Futhermore, the overview of the repositories that followed the SLSA requirements will be presented.

6.2 Research Objectives

7 Research Plan

7.1 Phase One

Defining Unsafe Updates: Building on similar research conducted in the JavaScript ecosystem (https://ieeexplore.ieee.org/ the project's first phase involves defining what an unsafe update means within the context of Python and/or Java. Typically, an unsafe update could be one that introduces breaking changes, negatively affects performance, opens up security vulnerabilities, or adds incompatible API changes.

7.2 Phase Two

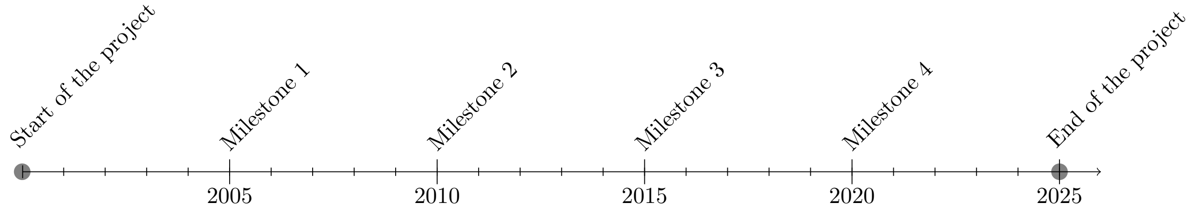
Implementation of Safety Checks: Next, we will extend the Macaron framework's functionality by implementing additional safety checks for these unsafe updates. Macaron is an extensible checker framework for supply chain security and CI/CD services, such as GitHub Actions. It allows adding new checks as Python modules and provides intermediate representations specifically designed for CI/CD services to facilitate verifying new properties.

7.3 Phase Three

Empirical Analysis of Real-World Projects: With the safety checks in place, the final phase of the project is an empirical study conducted on GitHub to ascertain the frequency of unsafe updates occurring in Python and/or Java projects. By understanding the 'how' and 'why' behind these updates, developers can adopt more informed, proactive strategies in their coding practices.

Research Stage	Deadline	Challenges
Data Collection	Month 1	Finding relevant sources
Examine Repositories	Month 2	Defining research questions

8 Timeline



Year	**Event**
2000	Start of the project
2005	Milestone 1
2010	Milestone 2
2015	Milestone 3
2020	Milestone 4
2025	End of the project

Table 2: Project Timeline

References

[1] OWASP Foundation. Owasp top 10 ci/cd security risks, 2023.

[2] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. Sok: Taxonomy of attacks on open-source software supply chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1509–1526. IEEE, 2023.

[3] Elias Levy. Poisoning the software supply chain. *IEEE Security & Privacy*, 1(3):70–73, 2003.

[4] Luis Mastrangelo, Luca Ponzanelli, Andrea Mocci, Michele Lanza, Matthias Hauswirth, and Nathaniel Nystrom. Use at your own risk: The java unsafe api in the wild. *ACM Sigplan Notices*, 50(10):695–710, 2015.

[5] Marcela S. Melara and Mic Bowman. What is software supply chain security?, 2022.

[6] U.S. Department of Defense. Defending continuous integration/continuous delivery (ci/cd) environments. PDF document, 2023.

[7] Sean Peisert, Bruce Schneier, Hamed Okhravi, Fabio Massacci, Terry Benzel, Carl Landwehr, Mohammad Mannan, Jelena Mirkovic, Atul Prakash, and James Bret Michael. Perspectives on the solarwinds incident. *IEEE Security & Privacy*, 19(2):7–13, 2021.