International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I1P108
Eureka Vision Publication | Volume 4, Issue 1, 62-74, 2023

Original Article

Secure Software Supply Chains in Open-Source Ecosystems

Sunil Anasuri
Independent Researcher, USA.

Abstract - Modern application development has been transformed by the rapid proliferation of open-source software, which enables agility, cost-effectiveness, and scalability through innovation. It has, however, also caused complex security problems in software supply chains. This paper takes a closer look at the changing threat environment that follows the open-source supply chain by covering the most noticeable avenues of attack (dependency confusion, typosquatting, and compromised maintainers). It examines real-world occurrences, such as hacks like SolarWinds and Log4Shell, as well as recent ecosystem hacks like PyPI, NPM, and GitHub, to demonstrate how attackers exploit the expectations of trust and the vulnerabilities of Pipelines to gain access to software production chains. We also assess the new defense conditions, such as Software Bills of Materials (SBOMs), the signing of artifacts, the hardening of the CI/CD pipeline, and tools of static and dynamic analysis. OpenSSF, SLSA, and Sigstore, as community-supported, policy-driven initiatives, are discussed in terms of how they support secure-by-design development methods. Kubernetes case studies and the best open-source repositories can provide valuable insights into effective risk mitigation and security implementation in practical systems. The paper offers a secure supply chain system that is specific to an open-source ecosystem and focuses on provenance verification, automation enforcement, and DevSecOps compatibility. Lastly, it touches on the existing constraints and important evaluation criteria, and future directions, proposing that every aspect of the ecosystem should cooperate to achieve resilient and trustworthy software. The objective of this research is to provide interested parties with the knowledge and tools that will help them combat the emerging threats in the supply chain.

Keywords - Software supply chain, Dependency management, SBOM, DevSecOps, SLSA, Sigstore, Kubernetes.

1. Introduction

Open-Source Software (OSS) has become the foundation of new software development, supporting a wide variety of applications, infrastructure components, and services. Its transparency, module-based system, and community development model have led to rapid innovation in cost reduction for development across the industry. These ubiquitous security issues, however, introduce increased alarm surrounding the security of the software supply chain. [1-3] Software supply chain comprises the full lifecycle of building the software and distributing it along with its dependencies, build tools, and deployment systems, many of which are open and decentralized ecosystems. Transparency, accessibility, and distributed trust are exactly the characteristics that make OSS useful, and they are simultaneously the vulnerabilities that it faces: confusion of dependency, typosquatting, injecting malicious code, and compromising maintainer accounts.

The SolarWinds breach and the Log4Shell vulnerability are recent examples of how trust can be undermined in software supply chains, and thus have attracted the attention of the global community. A single vulnerable dependency can spread system-wide risks to thousands of systems and organisations. These developments highlight the weaknesses of traditional security models in the open-source ecosystem, where many do not have centralised control, formal governance, or mandatory vetting procedures. There is therefore a pressing need to incorporate security across the entire software lifecycle, from secure coding practices and dependency checking through to signed packages and the secure distribution of artefacts.

The industry, in turn, is adopting proactive and verifiable security measures. There is growing popularity in techniques like creating Software Bills of Materials (SBOMs), signature hashing of components, continuous vulnerability scans, and reproducible builds. The desire to build resilient and transparent supply chains is being bolstered by initiatives, such as Sigstore, the Open Source Security Foundation (OpenSSF) and governmental requirements. The paper also explores the issues and challenges of OSS supply chain security and presents a roadmap to the adoption of best practices that can be used to ensure good security is attained without necessarily putting a strain on the collaborative process that is typical of open-source development.

2. Background and Motivation

The process of securing software supply chains has become a critical issue in the digital world, and increasingly, mainstream development activity is based on open-source software. [4-6] Context and motivation to improve supply chain security are given through background knowledge of the concepts in the basics of software ecosystems, the central element of open source, and

prominent attack examples on open source. The section elaborates on the key concepts and trends that underscore the significance of open-source supply chain security.

2.1. Software Supply Chain Concepts

A software supply chain refers to all the parties, events, and technologies which take part in generating, distributing and using software. Software supply chains are virtual (unlike physical supply chains) and are often opaque; they consist of a network of developers, third-party libraries, build tools, package registries, CI/CD pipelines, cloud platforms, and runtime environments. The supply chain is often divided into four fundamental steps: create, which involves writing source code and managing dependencies, build, which involves compiling and delivering code into deliverables, deploy, which involves moving artefacts into production facilities or users' systems, and run, when the software is used in live systems.

The phases are susceptible to varying forms of attacks. For example, vulnerable dependencies used during the creation stage may introduce covert vulnerabilities; malicious interventions in the build step may deliver malicious payloads; and unsecured deployment pipelines may lead to configuration drift or unauthorised modifications. These risks multiply as software ecosystems, enabled by the modular reuse of open-source components, become increasingly complex in the future. Cybersecurity repository statistics for 2023 indicate a 742% increase in software supply chain attacks over the previous three years, with 12% of data breaches resulting from such vectors, and an average cost of $4.63 million per breach. These disturbing numbers indicate the increasing popularity of supply chain attacks among threat actors and a subsequent need for countermeasures.

2.2. Role of Open Source in Modern Development

The software industry has been radically changed by open-source software. It enables international cooperation, rapid innovation, and low-cost development because the source code is freely available to anyone who wants to develop it. Tools such as GitHub, GitLab, and Bitbucket have made it extremely easy to handle open-source projects, allowing developers based in disparate regions of the world to contribute to collective codebases and even fix bugs and develop off one another. This has enabled the creation of robust, scalable software at an unprecedented speed.

Open-source components have become ubiquitous in nearly every domain today, including web development, mobile applications, machine learning, and enterprise software. According to the 2023 GitHub Octoverse report, 90% of the companies use open-source tools or libraries in their software stacks, and the number of contributions exceeded 413 million in 2022 alone. Nevertheless, this diffuse dependency brings some great risks. The vulnerability in a widely used open-source library can be cascaded to thousands of dependent software as the downstream dependencies are typically nested and not always known. Although open source promotes transparency and peer code review, its benefits are only convertible into security when paired with other formal security processes, such as dependency auditing, provenance checking, and auto-patching.

2.3. Motivating Examples of Attacks

2.3.1. SolarWinds Attack

The SolarWinds attack has been one of the most significant attacks on a software supply chain on record. Attackers who managed to access the build environment of the network monitoring platform Orion were high-tech. This trojan was secretly inserted into software updates and is known as malicious code, later referred to as SUNBURST, which was digitally signed to conceal its presence. These trojanized upgrades were subsequently sent to over 18,000 customers, including the U.S government agencies and Fortune 500 companies. Attackers have sustained unrecognized connection to victim systems and networks for months, as long as they are unnoticed. First reported in late 2020 and more thoroughly examined in 2023, the SolarWinds hack was an example of how malicious actors could exploit trusted software supply chains to disseminate malware on a large scale and circumvent standard perimeter defences.

2.3.2. Log4Shell Vulnerability

The vulnerability in the Apache Log4j project in Log4Shell (CVE-2021-44228) is another severe vulnerability that threatens the security of innumerable applications based on Java. Through this zero-day vulnerability, it was possible to execute remote code unofficially without authentication by injecting specially constructed log messages. The vulnerability was publicly disclosed in December 2021, leading to a worldwide hurry up to detect and fix vulnerable systems. Cybercriminals, such as botnet threats and ransomware operators, created and used exploits at an abnormally high rate. ZyXEL network devices and systems were among high-profile targets that were indirectly impacted via other Software packages using Log4j. Log4Shell highlighted again how a small, powerfully integrated open-source component can become a major point of failure within an expansive digital ecosystem to the point that continuous checkups, upgrades, and strong supply chain policies are required.

3. Threat Landscape in Open-Source Supply Chains

The open-source software supply chain has fallen under the radar of cyber attackers due to its distributed nature, complex dependency structures, and reliance on community-powered contributions. The threats exploit different sources of vulnerabilities at multiple points within the software lifecycle, through trust-based dependencies, and implemented automation systems that trade ease of use for security checks. [7-10] In this section, we discuss the most insidious attack vectors that currently endanger the open-source supply chain and look at the mechanisms through which vulnerabilities travel through the software ecosystem, occasionally to catastrophic effects.

3.1. Common Attack Vectors

The type of supply chain attack usually targeting open source environments typically attempts to inject malicious code into the environment or to exploit some trust mechanism to achieve unauthorized access or control. Some of the most indicative attack vectors on open-source supply chains include the following.

3.1.1. Dependency Confusion

Dependency confusion is a substitute attack, which is possible when internal software packages (which are meant to be used within an organization) are accidentally substituted with malicious versions of the same packages on publicly available repositories. Package managers, such as npm, PyPI, or NuGet, typically assume the most recent version available, even if it is publicly accessible. Automated build systems may pull down the attacker code upon finding a package with the same name as an internal one but a greater version; it can assume the name of an internal package without being under the same control. The attack vector became known publicly in 2021, when a researcher validated that dependency confusion could be exploited against numerous large enterprises, including Apple, Microsoft, and Tesla. This allowed the attacker to insert a harmless payload into corporate environments by publishing packages with internal names of that organization. This translated into an overlooked weakness in the package management system and a subsequent increase in awareness and implementation of reduction measures.

3.1.2. Typosquatting

Typosquatting is the deliberate use of the names of well-known or frequently used libraries (or other packages) in malicious packages. Such developers might download malicious code during installation by mistyping a package name, causing them to run malicious code. For example, running a version of the requests package instead of the official requests Python package may result in executing the code provided by the attacker. The attack is based on human error, and it is particularly effective in large-scale development environments, where people often share their scripts and configurations without conducting a fine-grained review of them. The typosquatting technique has also been widespread in various ecosystems, such as npm, PyPI, and RubyGems, and has frequently been used to facilitate data theft, cryptocurrency mining, or remote access to compromised systems.

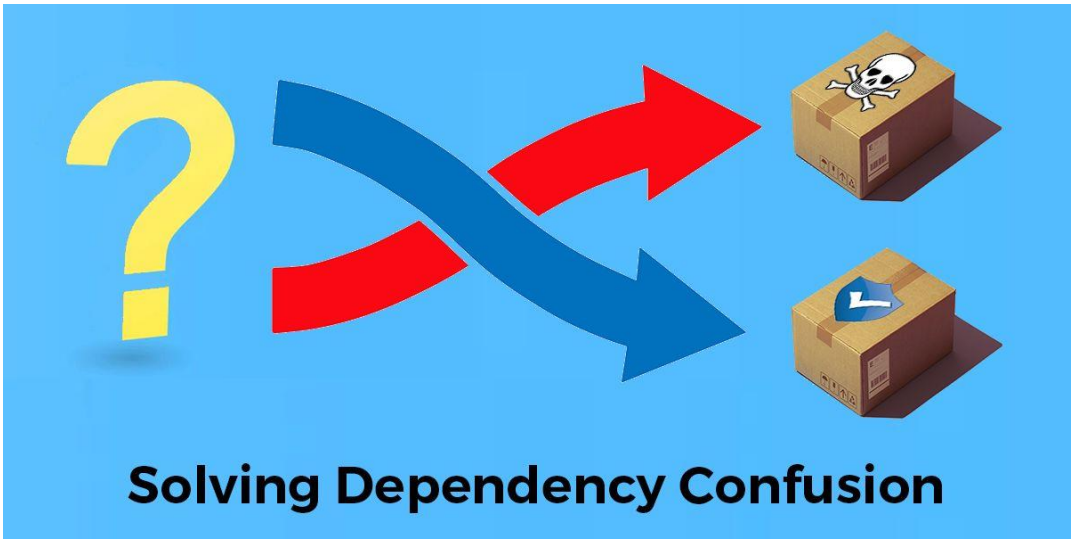

Figure 1. Dependency Confusion

Figure 2. Typosquatting

3.1.3. Compromised Maintainers

A small group of volunteers, or even just one person, tends to maintain an open-source project. Once an attacker has the account of a maintainer or can trick them into adding malicious contributors, they get direct access to the source code, build processes, and release pipelines. Based on this point, an attacker can inject malicious code into new packages of legitimate code. This was the case during the infamous npm event-stream attack, where a malicious dependency intentionally added by a new maintainer ended up targeting a specific cryptocurrency wallet. In the same vein, in 2023, several projects on PyPI and npm were maliciously updated following the leak or theft of maintainers' credentials. These attacks highlight the importance of robust code reviews and effective governance practices within open-source communities.

3.2. Vulnerability Propagation

The interconnectedness of components is a key characteristic of open-source ecosystems. A single library can be used as a dependency in hundreds or even thousands of downstream projects, which creates a web of transitive dependencies. The resulting interconnectedness implies that security vulnerabilities introduced once may spread rapidly within the system and may not be confined to the periphery of the end user. To take an extreme example, a security bug in a utility library supporting a web framework may indirectly impact all applications developed with that framework, despite the framework developers never touching the vulnerable code, but rather those who made the application. The incident response and patching activities are complicated by such a chain reaction. A prime example of such propagation is the Log4Shell exploit, where millions of systems were exposed due to the vulnerability of the Log4j library at the end of dependency trees. Dependency tracking propagation is also magnified by the lack of standardized metadata and tooling support of those package ecosystems. Although tools such as SBOM (Software Bill of Materials) or dependency scanners enhance visibility, this is not a universal feature. Consequently, there is a chance that organisations may not realise they are at risk of using outdated or vulnerable packages until such exploitation is discovered.

3.3. Insider and Supply Chain Attacks

Insider and supply chain attacks pose a particularly distinctive and significant risk to open-source ecosystems, as they are difficult to detect and can circumvent typical security measures. [11-13] An insider attack is an attack conducted by people inside the organization or project, usually developers, maintainers or collaborators, that takes advantage of their privileged position to introduce bad code, leak sensitive information or sabotage software parts. Such an attack can prove so hazardous when used in cases where a sense of trust and shared management tends to substitute institutional regulation in an open-source society. Many projects may have general policies that include lax background checks, lack contributor vetting, and are susceptible to social engineering, credential theft, or even voluntary breaches by essential maintainers.

3.4. Attack Surface Mapping across the Supply Chain

Supply chain attacks are attacks that target third-party items, devices, or services used in the software development cycle. Such attacks work on the relationship of trust and dependency among the tools and actors within the chain. For example, a malicious hacker can exploit a common open-source library or a CI/CD system to gain indirect access to numerous downstream applications. SolarWinds and Codecov hacks are relevant examples of how hackers can exploit upstream components to gain access to highly secure environments. These attacks cannot be easily identified or countered due to their origin outside the direct control of the

victim organisation, often making them very hard to defend against by any of the more traditional perimeter-centred security models. This necessitates that end-to-end visibility, strong authentication and continuous monitoring are critical elements to help ensure risks from insiders and external interventions in the supply chain are minimized.

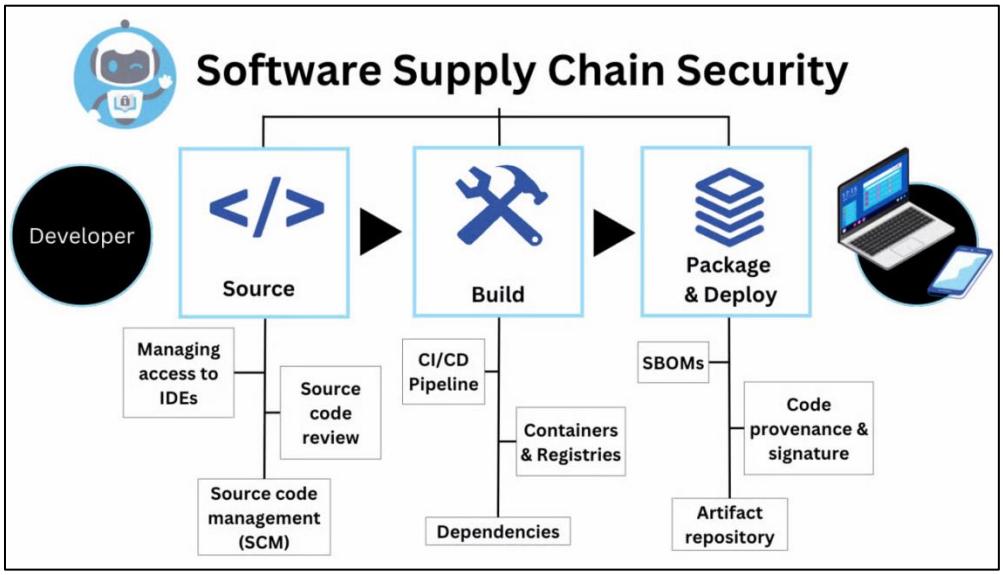

Figure 3. Stages of Software Supply Chain Security and Associated Security Controls

Hierarchical and proactive defence mechanisms must be present throughout the software development life cycle. It divides the process into four important stages: Source, Build, Package, and Deploy, which are all associated with a set of security controls that aim to mitigate the risks arising from their particular phase. In the Source phase, the process begins with secure developer environments and source code management, incorporating early intervention through code reviews and limited access to the IDE. Such steps are intended to identify weaknesses and stop improper instruments of code changes where they occur. Entering the Build, Package, and Deploy steps, the figure will showcase the incremental complexity and potential vulnerability points, especially in the case of including third-party dependencies. It promotes secure CI/CD pipelines, dependency scanning, SBOM creation, code signing, and protection of the artifact repository. It is possible to understand the significance of integrating the DevSecOps concepts throughout the supply chain by visualizing this end-to-end security architecture because the image highlights this point well. Finally, it can be used as an educational tool and a roadmap to implement, as it can help organizations know where to spend their security resources as a way to protect their businesses against the current risks to the supply chain.

4. Secure Development Practices

To address the supply chain issue, a shift-left security orientation is needed, which involves the implementation of security practices at the beginning [14-17] stages of software development and continuing to integrate security through the software life cycle.

4.1. Code Integrity and Provenance

Code integrity and provenance are the basis of secure development. Code integrity deals with a check on whether the source code and other artifacts have been compromised, amongst provenance, which deals with where the code came from, who wrote the code, and when the code was written. These objectives are normally realized by cryptographically signing commits, tags, and software artifacts so that any malicious change is recorded. A more trustworthy development history is achieved through signed commits, e.g., required by GPG or Sigstore in a Git-based system. Provenance tracking, on the other hand, allows developers and security teams to opine on the trustworthiness of a codebase by inspecting its genealogy and checking the identity of the contributors. Provenance is also key in determining the risk of implementing internal or external code in an open-source ecosystem since almost any actor can contribute code elements.

4.2. Dependency Management and SBOM (Software Bill of Materials)

Dependency management is a crucial aspect to consider with modern applications that include dozens or even hundreds of third-party libraries. Developers need to assess the security state of third-party packages, avoid using deprecated or discontinued components, and continually analyse threats within the software lifecycle. Tools such as Dependabot, Renovate, or OpenSSF

Scorecards help automate dependency updates and analysis. The development and utilization of a Software Bill of Materials (SBOM), A broad inventory that contains an enumeration of all components, libraries and dependencies in a work, is an important part of this process. SBOMs make it easy to detect vulnerable packages in a short span of time in case a new exploit is announced, speeding up the response significantly. SBOMs are becoming a common security requirement as regulatory bodies and software consumers increasingly call for the inclusion of SBOMs to achieve transparency and manage supply chain risks.

4.3. Vulnerability Scanning and Static Analysis

Vulnerability scanning and Static Application Security Testing (SAST) are used to identify potential problems in advance, providing security during development. Vulnerability scanners analyse code, both first-party and third-party, seeking pre-existing vulnerabilities, configuration errors, or unsafe usage patterns. Static analyzers examine the source code, or more commonly program intermediate representations, without running the program and can identify bugs like buffer overflows, injection points and unsafe API calls. Static tools, such as SonarQube, Semgrep, and CodeQL, automate scanning and can be embedded directly into development environments or CI/CD pipelines. The safety ensured by regular scanning is not only that the window of opportunity is shut down, but also the de facto inability to forego the security as an activity worth repeating. Introducing them into the development process will help teams to stay in synchronization with code modifications because the security checks will be made as the teams work on their problems, thus mitigating the number of vulnerabilities that enter the production environment.

4.4. CI/CD Pipeline Hardening

The Continuous Integration and Continuous Deployment (CI/CD) pipeline is the backbone of contemporary DevOps pipelines, and at the same time, one of the richest targets for attackers. Securing the tools, configurations and credentials used to power automated builds and deployment is hardening the CI/CD pipeline. Important activities include applying least-privilege access, separating build environments, and validating inputs and outputs at all levels. The management of secrets is essential- tokens and credentials must be encrypted and should never be visible in plain text in repositories. Additionally, lateral movement is facilitated by secure execution environments (e.g., ephemeral runners or sandboxed containers), which make lateral movement more difficult in the event of a compromise. Pipeline logging and monitoring facilitate easier detection of aberrations, whereas security checks included in the CI/CD life cycle (e.g., SAST, SBOM validation) identify vulnerabilities earlier, before they enter production. On the whole, a hardened CI/CD pipeline changes automation as a possible attack vector into a strong security control.

5. Policy, Governance, and Community Initiatives

The technical response is no longer adequate as the threats to the software supply chain continue to increase. An effective governance system, which includes policies and the community, as well as cross-organizational cooperation, is needed to enhance trust and security in the open-source ecosystem. [18-20] Such initiatives codify best practices, but also encourage a shared responsibility model, where participants, maintainers, organizations and governments all have a collective role to play in protecting the supply chain.

5.1. OpenSSF, SLSA, and Sigstore

Several community-based programs have emerged to address the challenges of supply chain security. The Open Source Security Foundation (OpenSSF) is a major consortium within the Linux Foundation, with a mission to enhance the security of open-source software by uniting industry stakeholders in partnership through working groups and projects. One of its more notable contributions is the supply-chain levels for software artifacts (SLSA), a model of security by designating successive tiers of assurance over software build integrity. SLSA provides verifiably sound statements regarding software artefact construction and promotes good construction practices, such as hermetic builds and build provenance. Sigstore is a similar project of the OpenSSF that streamlines cryptographic signing and verification of open-source artefacts. It allows developers to code-sign container images and binary data using short-lived, identity-based keys. In combination, these tools and frameworks provide a pragmatic and scalable foundation for enhancing transparency, traceability, and trust in software projects.

5.2. License Compliance and Risk Management

Technical vulnerabilities and governance should also address legal and compliance risks. Open-source licenses are terms that allow open-source code to it can be developed and redistributed. In the event of non-compliance with license conditions, it may expose a developer to litigation proceedings, public backlash, or even coercion to remove the code. Organizations must have a current inventory of all the parts of the software, often through SBOMs, and map each to the obligations of the license associated with it. Automatic license scanners (e.g., FOSSA, ScanCode and Clearly Defined) can facilitate the process of identifying license types and probable conflicts. Risk management implies reviewing the state of the health and maintenance of dependencies. A riskier project is not being maintained or has no responsible disclosure. A standard governance policy, such as vetting of new dependencies, monitoring package health, and well-defined escalation procedures, should be followed in the event of a

vulnerability or license violation. Such a proactive strategy will ensure that the use of open-source software is secure, compliant, and sustainable on a large scale.

5.3. Ecosystem-Wide Collaboration

Open-source development is highly complex and decentralized, and this nature requires collaboration across the ecosystem to mitigate the risks in the supply chain. This involves collaboration among open-source maintainers, security researchers, vendors, academics, and government organisations. Collective databases of vulnerability, such as CVE and OSV, as well as coordinated vulnerability disclosure processes, enable the community to react quickly and harmoniously to threats.

Regulatory and government agencies are also increasing their contributions. As an example, the U.S. Executive Order 14028 on Improving the Nation's Cybersecurity requires the use of SBOM and secures development guidelines in relation to software procurement by the federal government. These top-down measures promote the spread of best practices in supply chain security. Community funding and support for Critical projects in open-source infrastructure, such as those in programs like the OpenSSF Alpha-Omega or GitHub Sponsors, can help support infrastructure projects with the resources they require to implement necessary security changes. After all, the only way any software supply chain can be made more secure and resilient is through collaborating across organizational and sectoral lines, and that is what the open-source community can provide.

6. Case Studies and Real-World Examples

To understand the difficulty areas of securing open-source software supply chains, it is necessary to consider real-life incidents. The variety and gravity of threat examples are showcased in the case studies of ecosystems (e.g., Kubernetes, Python (PyPI), JavaScript (NPM), and GitHub CI/CD). All these instances highlight the potential of exploits, attackers, and security incompetence to cause major breaches to developers, companies, and users worldwide.

6.1. Securing Kubernetes

Modern cloud-native applications are built on the foundational infrastructure called Kubernetes, which has become a prime target for supply chain attacks. The Red Hat State of Kubernetes Security Report 2023 discovered that almost two-thirds of the surveyed companies put off deployments because of their security concerns. Targeted vulnerabilities in application components (32%), lax controls in the CI/CD pipeline (30%), and a lack of Software Bills of Materials (SBOMs) or code provenance (29%) were identified as the most substantial causes. The number of security incidents and misconfigurations has also been sharply increasing during the runtime stage, indicating the necessity of further incorporating DevSecOps. Kubernetes clusters should not be the only ones to be secured during runtime; the entire software lifecycle, including image builds and deployment pipelines, needs to be protected from manipulation and even human error.

Table 1. Kubernetes Security Trends

Delayed Deployments	Runtime Incidents	Vulnerabilities Remediated	Misconfiguration Incidents
67%	49%	42%	45%
55%	30%	38%	53%
55%	32%	31%	59%

These statistics indicate the increasing awareness that Kubernetes security is no longer all about the safety of clusters at runtime but about keeping upstream artifacts and the supply chain transparent.

6.2. Python/PyPI and NPM Supply Chain Incidents

Malicious package managers, such as PyPI and NPM, used exclusively by open-source package managers, are gaining momentum. An attack on the PyPI repository in 2023 lasted for one year and was carried out by malicious packages posing as AI chatbot tools. Though it only hit about 1,700 downloads, the campaign crossed 30+ unique packages and targeted developers across more than 30 countries, indicating the advanced preparedness of attackers who currently plan attacks on long-tail, more specific intrusions.

Table 2. PyPI and NPM Supply Chain Incidents

Incident	Affected Packages	Downloads/Exposure	Attack Method
PyPI AI Chatbot Malware Campaign	30+	1,700+ downloads	Malicious package uploads
NPM RED-LILI Automated Packages	800+	Hundreds of projects	Automated dependency confusion
NPM Gluestack Compromise	17	1M+ weekly downloads	Remote access trojans

In the meantime, the JavaScript/NPM infrastructure experienced a significant increase in automated malicious uploads. REDS-LILI, an attack that used automated dependency confusion and package flooding, published more than 800 malicious packages, affecting hundreds of unknowing projects. The other blatant hack was that of 17 packages of Gluestack, which had over 1 million weekly downloads and released remote access trojans that had exfiltration capabilities.

6.3. GitHub and CI/CD Exploits

Modern software development has become GitHub-centred, particularly after so-called GitHub Actions became effectively built-in to many CI/CD pipelines. This popularity has attracted its enemies, who exploit misconfigurations and assumptions of trust in automating pipelines. In 2023, a team of researchers found a new attack channel against self-hosted GitHub Actions runners. Attackers could exploit a lack of proper isolation in pipelines by using pull requests to inject malicious code, allowing them to run arbitrary commands with elevated privileges. The next major breach was that of the famous GitHub Action tj-actions/changed-files, deployed in more than 23,000 repositories. Hackers penetrated the system to steal secrets, such as AWS keys and GitHub tokens, by logging the information. This incident demonstrated that even a single compromised code insertion/delivering mechanism can lead to large-scale data leakage, particularly when secrets are not properly controlled or poorly secured.

Table 3. GitHub CI/CD Exploit Incidents (2023)

Incident	Affected Repositories	Impacted Assets	Attack Vector
Self-Hosted Runner Exploit	Thousands	CI/CD pipelines	Malicious code injection
tj-actions/changed-files Breach	23,000+	Secrets (keys, tokens)	Compromised GitHub Action

These illustrate the importance of auditing third-party activity, applying best practices for secrets management, and hardening CI/CD environments to make them resistant to undesirable tampering by insiders or other parties.

7. Proposed Secure Supply Chain Framework

7.1. Architecture Overview

Hierarchical structure that covers the whole lifecycle of an open-source software development and deployment. It separates the process into key zones, including Open Source Contribution, Pre-Build Security, CI/CD Pipeline Security (Build Zone), Governance & Policy Layer, and Deployment & Runtime. Each zone demonstrates the applicable tools, checkpoints, and stakeholders that ensure the code flowing through the pipeline is trusted, verified, and secure. Within the Open Source Contribution Zone, developers will commit code to a Git-based repository, such as GitHub or GitLab. Basic quality and policy requirements are verified by the automated pull request (PR) checks and human code reviews. When it passes validation, it is promoted to the Pre-Build Security Zone, where static code analyzers, SBOM generators, and dependency scanners can be interviewed. These scan the source code, produce metadata regarding dependencies, and generate a bill of materials that is referable downstream. The most important state of the security architecture is the CI/CD Pipeline Security (Build Zone). In this case, orchestrators like GitHub Actions or Jenkins trigger builds, and they use a safe environment to fulfil the criteria of the SLSA Level 3+. Build artifacts are then signed with tools such as Sigstore or Cosign, and deposited in secure artifact repositories such as JFrog or Nexus.

This makes it tamper-proof and traceable as metadata is recorded and outputs are signed. The signed artifacts are then published, and provenance and integrity testing are integrated into the deployment process. Lastly, the Deployment and Runtime Zone is where artifacts are used through public registries such as NPM, PyPI, or Maven. The packages are also validated using SBOM validators and signature verifiers before being communicated to the end-user or target system, ensuring that only trusted packages are deployed. Then, systems such as Falco can be used to monitor the runtime behavior and raise alerts when something is not normal or a policy violation. The strength of this full-stack solution lies in the fact that immutable audit logs, combined with Governance and Policy Layer checks regarding legal/license compliance, ensure that the entire supply chain is defensible as a cohesive system that can be easily adjusted to current DevSecOps processes.

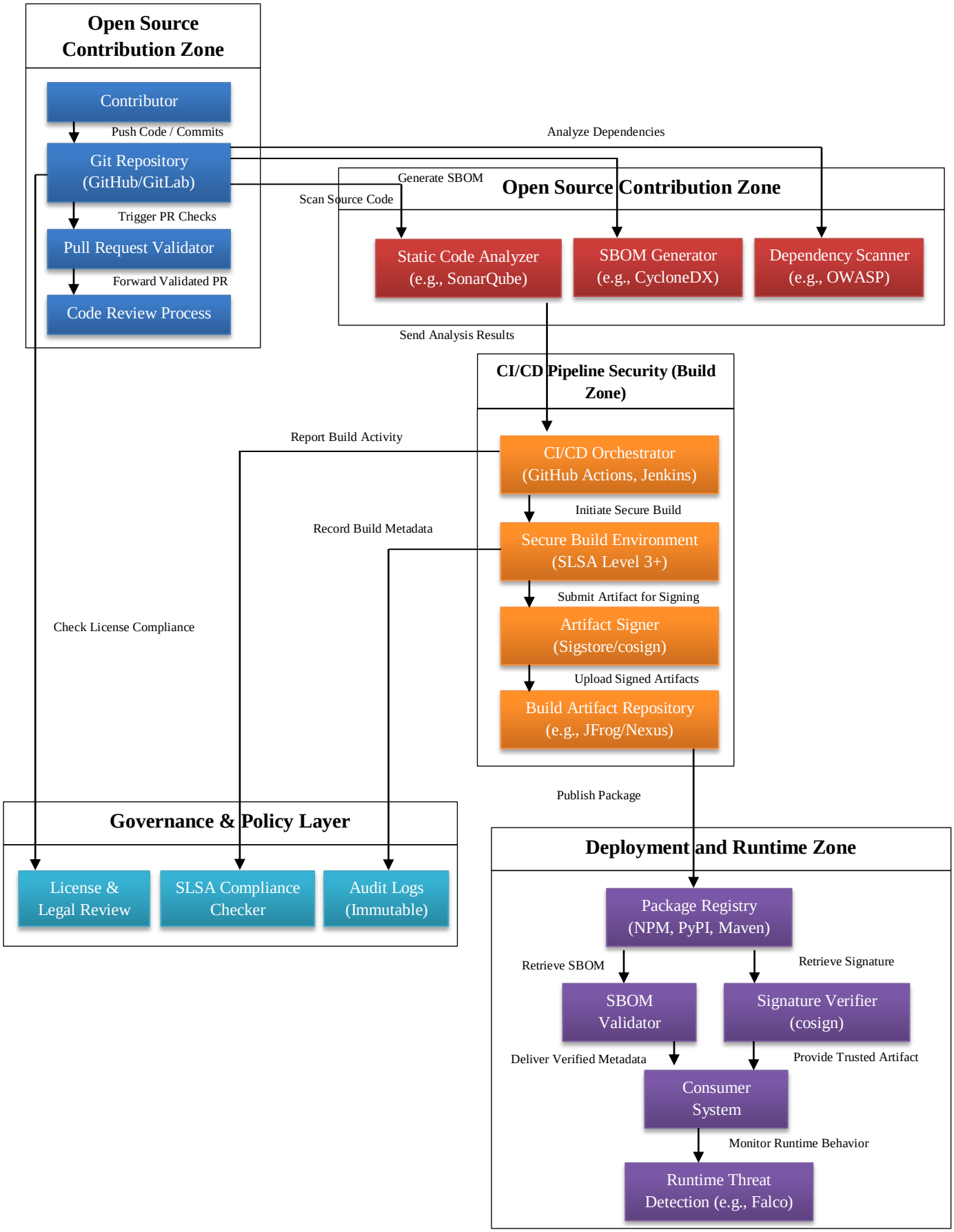

Figure 4. End-to-End Secure Software Supply Chain Architecture for Open-Source Ecosystems

7.2. Security Enforcement at Each Stage

To ensure a secure supply chain, a strong framework requires a multi-faceted security to be implemented across the software lifecycle, right from the commencement of code contribution up to the endpoint of the deployment stage. The initial step of enforcement in the source contribution phase involves limited controlled access to repositories for developers, required validations of pull requests, and human code reviews. These guarantee that only authorized contributors are allowed to edit the codebase and that the changes submitted are reviewed both automatically and manually.

When code reaches the pre-build security zone, it uses tools like static code analyzers, software bill of materials (SBOM) generators and dependency scanners to require vulnerability disclosure prior to compilation. This proactive assessment minimizes the chances of malicious or susceptible components being installed in the application. Then the CI/CD build pipeline comes into play, where secure environments are implemented to ensure build integrity. All produced artifacts, are digitally signed, so final tampering could be detected. Building metadata is also logged in artifact stores, which makes tracking of provenance and rollback on compromised build something artifact stores can support.

The deployment and runtime stage carries on the enforcement with signature checks and validation of the SBOM. The authenticity of the artifacts is checked before they are fetched into the package registries such as NPM or PyPI. Lastly, run-time threat detection technologies like Falco watch the behavior of systems, or policy violations or suspicious activity and make sure that even after the attacker may have successfully bypassed previous defences, they are spotted on production systems.

7.3. Integration with DevSecOps Practices

The proposed framework should be entrenched in the processful DevSecOps workflows, so as to be effectively scalable; security enforcement must be based on the principles of automation, agility, and continuous delivery. DevSecOps focuses more on left-shifting security, which involves integrating security early and continuously into the development process, rather than making it an afterthought. This is supported by a secure supply chain architecture that embeds tools such as SBOM generators and static analyzers into the developer workflows and CI/CD pipelines.

Automation is an essential factor. Services such as GitHub Actions or Jenkins can perform automated security testing and signing of artifacts, and policy enforcement tools ensure the software license compliance and rule execution without direct manual action. This enables teams to have a fast deployment schedule without sacrificing security. In addition, immutable and secure audit logs that build records enhance traceability and reduce the time to respond to critical incidents that meet current compliance expectations. Finally, feeding the production environment with runtime threat detection and behavioural analytics enables development environments to create feedback loops. The feedback loop of DevSecOps can be closed through insights gained from runtime anomalies, which can be leveraged in future code reviews and dependency decisions. Through this, the framework can accommodate a culture in which development, operations, and securities teams work effectively to strike a balance between velocity and integrity in software delivery.

8. Evaluation and Discussion

8.1. Metrics for Supply Chain Security

To analyze the security of any software supply chain, a list of quantitative and qualitative performance measures that also indicate the level of compliance, along with the level of resiliency and responsiveness to threats, is needed. The most typical measures are the presence of a Software Bill of Materials (SBOM), the quantity and criticality of unaddressed vulnerabilities, the Time To Patch (TTP), and the effectiveness of signing and verification artifacts. Mean Time to Detect (MTTD) and Mean Time To Respond (MTTR) to incidents are also crucial indicators of operational efficiency. Other critical metrics are reporting of both static and dynamic analysis tools, utilization of authenticated sources of dependencies and runtime incidence rate. The metrics will enable stakeholders to assess the maturity level of their supply chain security and identify areas that require attention or automation.

Table 4. Key Metrics for Evaluating Software Supply Chain Security

Metric	Description	Ideal Value/Status
SBOM Coverage	% of software artifacts with complete SBOM	100%
Vulnerability Remediation Rate	% of known vulnerabilities fixed within SLA	>90%
Artifact Signing Rate	% of build artifacts cryptographically signed	100%
TTP (Time to Patch)	Average time to apply critical patches	<7 days
MTTD/MTTR	Time to detect/respond to security incidents	<24 hrs detection, <48 hrs response

Dependency Source Trust Level	Use of verified/official sources for third-party components	High
Runtime Threat Incidents	Number of detected runtime anomalies per release	Near-zero or decreasing trend

8.2. Comparison of Tools and Techniques

Various tools and techniques have emerged to address different stages of the software supply chain, including source code analysis, artefact verification, and execution monitoring. Both tools have particular advantages and are accompanied by trade-offs concerning integration complexity, cost and automation capacities. Specifically, SonarQube is great at static code analysis and identifying bugs at an early stage. In contrast, CycloneDX is excellent at generating SBOMs in a rich format, facilitating transparency and helping to trace the components. Likewise, Sigstore/Cosign offers lightweight yet powerful cryptographic signing services for container images and build artefacts. However, tool performance is usually related to the level of DevSecOps practice maturity within the organization and the security status of the development pipeline. There are tools which help in compliance reporting (e.g. Licensee for compliance auditing of licenses) and there are tools that help in threat detection (e.g. Falco for runtime monitoring).

8.3. Limitations and Gaps in Current Approaches

Although there is a push towards improvement in the security of supply chains, existing strategies have certain gaps and weaknesses. A significant constraint is that there is no standardization when it comes to the production, consumption, and interpretation of SBOMs among tools. SBOMs are also limited by inconsistent integration into CI/CD pipelines, which hinders real-time validation, even in cases where they exist. Furthermore, although artefact signing is gaining popularity, not every registry performs signature checks during the package pull operation, which reduces the protective value of such a signature. The next difficulty is tracing and policing in hybrid or dispersed development spaces. Most of the small- and medium-sized companies do not have the savvy or the capability of setting up security tools in an effective manner, leaving pipelines vulnerable. Moreover, the majority of vulnerability scanners are based on known CVEs and are unlikely to detect high-quality threats, such as zero-days or logic bombs through dependency confusion.

9. Future Directions

The future of security in supply chains is an adaptive and verifiable mechanism of trust that is automated to address the increasing complexity and interdependency of software ecosystems. New standards such as SLSA (Supply-chain Levels for Software Artifacts) and in-toto attestations are opening the door to tamper-evident build processes and policy-enforceable (machine-readable) provenance verification. These are evolutions that aim to transfer trust in properties ensured by manual checks and outside auditors into trust in cryptographic checks made on software artifacts throughout the pipeline. Furthermore, the introduction of AI-based analysis of threats and behavioral baselining will increase the capacity to identify unusual activity in the development and deployment settings.

Additionally, it is expected that the future will bring an even higher level of ecosystem-level cooperation, where governments, open-source foundations, and industry members will converge around common compliance frameworks, transparency requirements (such as mandatory Software Bill of Materials, or SBOMs), and secure-by-design standards. The OpenSSF, the EU Cyber Resilience Act, and U.S. Executive Orders on Cybersecurity indicate that the future supply chain security can no longer become optional; it will be regulated and complexified by liability incentives. Software supply chain security will not only be critical to software integrity, but also to national and economic security if we turn software into infrastructure.

10. Conclusion

The safety of open-source software supply chains has become a great concern in the contemporary digital world. The attack surface keeps growing as a result of more software development being dependent on third-party libraries, CI/CD automation, and shared repositories. The low-level breaches in greatly publicized events within the ecosystems such as PyPI, NPM, Kubernetes, and GitHub have demonstrated how simple vulnerabilities in complex Chief Execution Systems would ultimately lead to high-level breaches. End-to-end supply chain security, ensuring it via code integrity checks, SBOM implementation, secure builds, and sophisticated governance, is no longer a best practice, but a mandatory requirement.

Although significant changes have been achieved through projects such as OpenSSF, Sigstore, and SLSA, much still needs to be done. STIL, checking of trust automation is still in development stages and enforcing security across different toolchains in a harmonized way. The industry needs to embrace the notion that security is a collaborative responsibility to ensure we can move forward where all the stakeholders of the ecosystem (developers, maintainers, platforms and consumers) work together to prioritize

transparency, automation and resilience. It is only with a consolidated approach that we can begin to secure the software that supports our most vital systems and infrastructures.

References

[1] Boehmke, B. C., & Hazen, B. T. (2017). The future of supply chain information systems: The open source ecosystem. Global Journal of Flexible Systems Management, 18(2), 163-168.

[2] Sahay, B. S., & Gupta, A. K. (2003). Development of software selection criteria for supply chain solutions. Industrial Management & Data Systems, 103(2), 97-110.

[3] Haulder, N., Kumar, A., & Shiwakoti, N. (2019). An analysis of core functions offered by software packages aimed at the supply chain management software market. Computers & Industrial Engineering, 138, 106116.

[4] Von Krogh, G. (2003). Open-source software development. MIT Sloan Management Review.

[5] Wolff, E. D., GroWlEy, K. M., Lerner, M. O., Welling, M. B., Gruden, M. G., & Canter, J. (2021). Navigating the SolarWinds supply chain attack. Procurement Law, 56, 3.

[6] Ladisa, P., Plate, H., Martinez, M., & Barais, O. (2022). Taxonomy of attacks on open-source software supply chains. arXiv preprint arXiv:2204.04008.

[7] Li Wenyan, Liu Fang, Zhao Feng, et al. "Analysis method of open source components of package-free management files based on vulnerability mining." Software Engineering and Application, 2020, 49(10): 86-91.

[8] Kilamo, T., Hammouda, I., Mikkonen, T., & Aaltonen, T. (2012). From proprietary to open source—Growing an open source ecosystem. Journal of Systems and Software, 85(7), 1467-1478.

[9] Guo Ming, Zhang Lin. "Review of version tracking and determination technologies for open source components of package-free management files." Computer Application Research, 2019, 36(11): 3218-3225.

[10] Wang Yafei, Zhou Yunfei, Zhang Yu. "A method of open-source component identification based on code static analysis." Computer Engineering and Design, 2018, 39(12): 3142-3147.

[11] Miller, J. F. (2013). Supply chain attack framework and attack patterns (No. MTR140021).

[12] Ellison, R. J., & Woody, C. (2010, January). Supply-chain risk management: Incorporating security into software development. In 2010, 43rd Hawaii International Conference on System Sciences (pp. 1-10). IEEE.

[13] Checkmarx Finds Threat Actor 'Fully Automating' NPM Supply Chain Attacks, securityweek, 2022. online. https://www.securityweek.com/checkmarx-finds-threat-actor-fully-automating-npm-supply-chain-attacks/

[14] Imran, M., Hlavacs, H., Haq, I. U., Jan, B., Khan, F. A., & Ahmad, A. (2017). Provenance based data integrity checking and verification in cloud environments. PloS one, 12(5), e0177576.

[15] Ahmed, Z., & Francis, S. C. (2019, November). Integrating Security with DevSecOps: Techniques and Challenges. In 2019 International Conference on Digitization (ICD) (pp. 178-182). IEEE.

[16] Li, J. (2020). Vulnerabilities mapping based on OWASP-SANS: a survey for static application security testing (SAST). arXiv preprint arXiv:2004.03216.

[17] Li Ting, Hu Dongsheng, Zhang Baoqing, et al. "Study on the determination method of open source components of package-free management files." Computer Science and Exploration, 2017, 11(11): 1425-1433.

[18] Peng Zhang, and Haiyan Liu. "Summary of open source components for package-free management files." Computer Engineering, 2014, 40(8): 136-140.

[19] Ladisa, P., Plate, H., Martinez, M., & Barais, O. (2022). "Taxonomy of Attacks on Open-Source Software Supply Chains." arXiv preprint arXiv:2204.04008.

[20] Arundel, J., & Domingus, J. (2019). Cloud Native DevOps with Kubernetes: building, deploying, and scaling modern applications in the Cloud. O'Reilly Media.

[21] Pappula, K. K., & Rusum, G. P. (2020). Custom CAD Plugin Architecture for Enforcing Industry-Specific Design Standards. *International Journal of AI, BigData, Computational and Management Studies*, *1*(4), 19-28. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I4P103

[22] Rahul, N. (2020). Vehicle and Property Loss Assessment with AI: Automating Damage Estimations in Claims. *International Journal of Emerging Research in Engineering and Technology*, *1*(4), 38-46. https://doi.org/10.63282/3050-922X.IJERET-V1I4P105

[23] Enjam, G. R., & Chandragowda, S. C. (2020). Role-Based Access and Encryption in Multi-Tenant Insurance Architectures. *International Journal of Emerging Trends in Computer Science and Information Technology*, *1*(4), 58-66. https://doi.org/10.63282/3050-9246.IJETCSIT-V1I4P107

[24] Pappula, K. K., & Rusum, G. P. (2021). Designing Developer-Centric Internal APIs for Rapid Full-Stack Development. *International Journal of AI, BigData, Computational and Management Studies*, *2*(4), 80-88. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V2I4P108

[25] Pedda Muntala, P. S. R., & Karri, N. (2021). Leveraging Oracle Fusion ERP's Embedded AI for Predictive Financial Forecasting. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(3), 74-82. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P108

[26] Rahul, N. (2021). Strengthening Fraud Prevention with AI in P&C Insurance: Enhancing Cyber Resilience. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(1), 43-53. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I1P106

[27] Enjam, G. R., & Chandragowda, S. C. (2021). RESTful API Design for Modular Insurance Platforms. *International Journal of Emerging Research in Engineering and Technology*, *2*(3), 71-78. https://doi.org/10.63282/3050-922X.IJERET-V2I3P108

[28] Rusum, G. P., & Pappula, kiran K. . (2022). Event-Driven Architecture Patterns for Real-Time, Reactive Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(3), 108-116. https://doi.org/10.63282/3050-922X.IJERET-V3I3P111

[29] Pappula, K. K. (2022). Architectural Evolution: Transitioning from Monoliths to Service-Oriented Systems. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 53-62. https://doi.org/10.63282/3050-922X.IJERET-V3I4P107

[30] Jangam, S. K., & Karri, N. (2022). Potential of AI and ML to Enhance Error Detection, Prediction, and Automated Remediation in Batch Processing. *International Journal of AI, BigData, Computational and Management Studies*, *3*(4), 70-81. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I4P108

[31] Pedda Muntala, P. S. R. (2022). Anomaly Detection in Expense Management using Oracle AI Services. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(1), 87-94. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P109

[32] Rahul, N. (2022). Optimizing Rating Engines through AI and Machine Learning: Revolutionizing Pricing Precision. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(3), 93-101. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I3P110

[33] Enjam, G. R., & Tekale, K. M. (2022). Predictive Analytics for Claims Lifecycle Optimization in Cloud-Native Platforms. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(1), 95-104. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P110