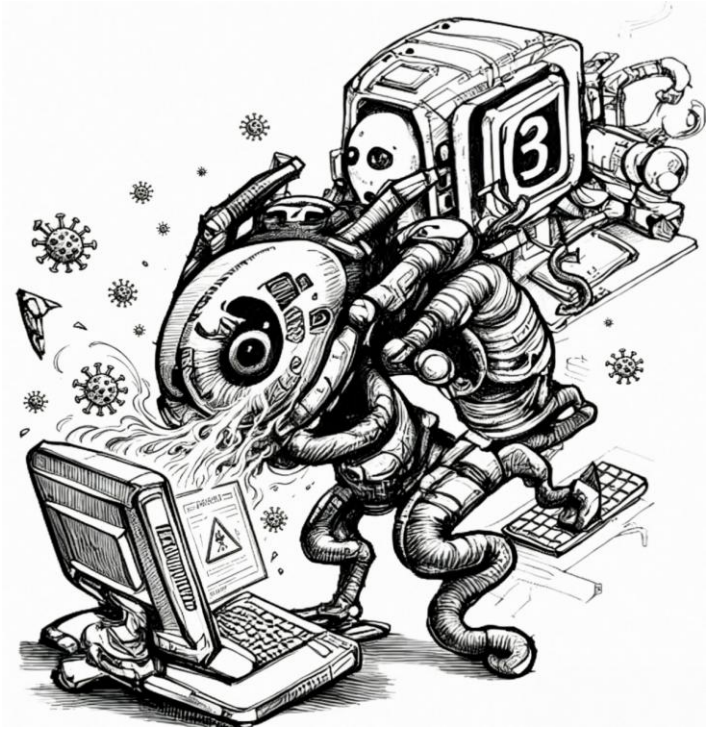
Sand Dune Collapse in the AI Era

-- Supply Chain Security Challenges under the TeamPCP Attack Model

Antiy CERT



First draft completed: June 5, 2026

First published: June 8, 2026

Update time of this edition: June 9, 2026

Table of Contents

Overview

Antiy CERT pointed out in its report "Sandstorm-style Poisoning Targeting Developer Tool Supply Chains - Sample, Technical, and Tactical Analysis of TeamPCP Organization" that the organization is different from the traditional hidden supply chain attack method, and has constructed a "sandstorm" poisoning paradigm, which excavates ecological safety cracks through full-chain batch poisoning, thus realizing large-scale invasion and rapid realization of black production. This article continues to sort out the impact of AI on cyber attacks as shown by the TeamPCP sandstorm poisoning incident. Combing the security risks of the tool supply chain ecology, trying to sort out the open source-based software tool chain ecology in the era of artificial intelligence, presenting its panoramic risks, combing the list of exposed surfaces, and analyzing the hierarchical transmission of supply chain threats. Designed to help developers fully understand the risks and make targeted adjustments and improvements.

In the past, CERT's analysis of supply chain threats focused on the perspective of threat samples, attack tactics and attack organizations. The current severe poisoning attack situation requires us to switch to the complete scene understanding dimension to promote security capability upgrade. Compared with the mature development ecology, there is still a certain gap in our overall understanding of the full link tool chain and the operating environment. Antiy CERT strives to carry out independent exploration and output practical results at the intersection of development ecology and safety protection. However, due to the lack of familiarity in this cross-border field, there are inevitably omissions and biases in this research content. We sincerely welcome criticism and correction from industry colleagues.

1 AI has become the “driver” of “sandstorm-style” supply chain poisoning.

Conventional software supply chain attacks are usually long-term latent and directional penetration. Attackers are cautious and difficult to perceive. However, TeamPCP attack organizations adjust the attack methods to large-scale and batch poisoned "sandstorm" raids, but they can also implement adaptive load delivery based on different portals, quickly obtain a large number of breakthrough points and centrally harvest voucher-based information assets by polluting open source warehouse components in batches. This new type of attack has crossed the kill chain mode, and initially has the characteristics of intelligent killing network. The TeamPCP attack program completed multiple rounds of rapid iteration of Chalk/Debug, Shai-Hulud, Megalodon, and Mini Shai-Hulud in only 8 months. Each

round superimposes new technologies and attack ideas, relying on the full empowerment of AI. Attackers rely on the human-machine collaboration of the AI system to analyze attack portals, write and debug malicious code, bypass defense rules, and operate the attack system, shortening the tool evolution cycle from monthly to weekly and daily levels.

1.1 Fully leveraging large models to generate malware

When developing Mini Shai-Hulud worm, TeamPCP uses human-computer cooperation mode to build malware technology stack. Claude 3.5 Sonnet and Claude Code CLI are used as the main tools to generate project scaffolding, startup scripts and backdoor implantation codes with one click. Relevant operations leave complete evidence through warehouse labeling and exclusive submission email. The organization is complemented by GPT-4o to optimize attack logic, write multiple layers of obfuscation-free code, and complete code snippets with the help of GitHub Copilot. The organization did not deliberately remove the traces of AI generation, the whole process by the large model to undertake the code engineering implementation work. High-order attack links such as CI/CD poisoning and signature forgery are designed manually. Relying on the big model, the organization significantly reduced the cost of writing malicious code and created a reusable supply chain attack template. The template was subsequently spread by the black circle, and some third parties also used the local open source code model to make secondary variants of malicious programs.

Table 1 A list of large model tools used by TeamPCP organizations1

Technology Classification	Model/Tool Name	**Tool specific version**	Undertaking development work	Confidence level	Corroboration source
Cloud Core Generation Model	Anthropic Claude	Claude 3.5 Sonnet Claude Code CLI	1. The whole set of project directory scaffolding, Bash startup script batch generation. 2. README, embedded Markdown format notes, redundant description text writing 3. Write backdoor code for. claude/settings.json persistence 4. Multi-file batch initialization, CI configuration template generation.	100%	Attacker GitHub warehouse marked vibe coded;Git submits email claude@users.noreply .github.com;OX Security, CSA, Antiy AVL Code Automated Analysis Report

Cloud-assisted iterative model	OpenAI GPT-4	GPT-4o / GPT-4 Turbo	1. OIDC token theft, pipeline permission exchange core logic refinement 2. JS multi-layer control flow flattening, Hex/AES-XOR double string obfuscation code 3. Bun runtime adaptation, npm/PyPI double package ecological adaptation branch development 4. EDR evasion, variable random renaming anti-killing code iterative optimization.	⩾90%	CSA Supply Chain Threat White Paper, OX Sample Disassembly, Tenable Attack Link Analysis Report
IDE Embedded AI Coding Tool	GitHub Copilot	GPT Series Homologous Base	Batch path traversal, environment variable enumeration, Git Actions YAML pipeline fragment completion; existing AI generate code fragments for secondary splicing and reuse.	70%	CSA sample per code snippet traceability analysis
Offline open source rewriting model (derivative variant)	CodeLlama	open source code model	Rewrite native loads offline, modify features to bypass cloud API traceability, and generate new variants in batches.	Confirm	SafeDep variant tracking report; only secondary modified samples after diffusion are used, TeamPCP the official original is not used
Offline open-source rewriting model (derivative variant)	DeepSeek-Coder	Open source code model	Localization adjustment obfuscation strategy, rewrite load loading logic	Confirm	The third-party black product is introduced during the second iteration of the original worm and does not belong to the attacker's initial development stack.

1.2 Attack operational efficiency has been significantly enhanced with AI assistance

The increase in attack efficiency brought about by the AI is not a steady small increase, but an explosive jump, and the TeamPCP series of attacks visually demonstrates this change in growth rate. The AI provides strong empowerment for attackers from the following links: First, the automatic reconnaissance capability can quickly capture platform metadata such as npm, PyPI and GitHub Actions, and efficiently screen high-value targets such as high-download dependent packets and high-frequency reuse CI/CD pipelines; Second, the automatic load generation mutation capability can produce malicious code variants with consistent functions and differentiated features in a

short time, stably avoid feature signature defense detection, and improve the success rate of attacks; third, the automatic distribution and implantation capability relies on the native capability of CI/CD platform to open up the whole link of target detection, penetration testing, supply chain poisoning and malicious diffusion, without manual intervention in the whole process, to ensure stable execution of attacks. The TeamPCP Megalodon action is a typical example of the fully automatic pipeline attack mode, and the whole poisoning process is completely separated from manual operation. Attackers rely on the automation system to greatly increase the attack execution efficiency, and a single batch action can achieve a very high percentage of intrusion success. In contrast to the traditional security operation center (SOC)'s stepped disposal mode of "alarm research and judgment-incident investigation-disposal response", the disposal circulation cycle is long, the interception success rate is limited, the execution efficiency gap between the attack and defense ends continues to widen, and the traditional defense rhythm cannot adapt to the attacker's automated attack mode of batch delivery and high success rate.

1.3 Large models empower the breakdown of trust systems and the disruption of provenance tracing.

In the Mini Shai-Hulud attack, the attacker hijacks the TanStack's official CI/CD pipeline, steals OIDC tokens, forges a trusted release credential that meets the SLSA L3 level, and completes an actual attack to break down the trusted standard. This is the first time that the system has been broken down. With a compliance endorsement, the malicious program has the same trusted identity as the regular component.

At the communication concealment level, the C2 domain name is disguised as a OpenTelemetry telemetry interface, and native channels such as GitHub submission, release and work orders are reused to transmit instructions and steal data, so as to cover up malicious behaviors through legitimate business traffic, greatly increasing the difficulty of traffic identification.

At the same time, the attacker uses AI batch generation traceability interference means: embedding Slavic folk keywords, character inversion encryption, multilingual mixed submission of comments, etc., to set up a "Russian roulette" damage load, deliberately disturbing traceability clues. Similar resources are clearly refined through the aggregation of large model platforms, which is typical of AI-enabled supply chain attacks, significantly raising the human and time costs of post-event traceability analysis.

1.4 Overall evolutionary trends of supply chain attacks

Combined with the attack and defense posture of TeamPCP and other cases, supply chain attacks have shown a multi-level evolution. In the attack mode, from the "precise and hidden" single-point accurate penetration to the "wide and fast" sandstorm batch delivery, after AI reducing the cost of evil, probabilistic large-scale attacks consume defense resources and cause alarm fatigue. In terms of combat form, from independent groups to ecological coordination of black and gray production, TeamPCP and other organizations are transformed into upstream threat service platforms, encapsulating attack capabilities into standardized resources, giving birth to "supply chain attack as a service" (SCAaaS), realizing commercialization and normal circulation. In the confrontation dimension, the attack target extends from system control to traceability confrontation and cognitive interference, and misleads attribution by implanting false clues and sharing source code. On the attack link, the entrance migrates to the upstream of CI/CD pipeline and cloud development environment, upgrading from terminal poisoning to source pollution. The popularization of AI development tools introduces cognitive layer attacks and implants malicious logic from the development root, thus putting pressure on the traditional defense system.

The macro trends and evolution characteristics of TeamPCP attacks are analyzed above. How these trends are implemented in a real attack link? The following figure shows the specific entities (accounts, platforms, package names, and downstream applications) involved in TeamPCP attacks and the attack paths between them in the form of entity-level threat association maps. The graph adopts a hierarchical architecture: the top layer is account credentials (GitHub account/PAT, npm Token, PyPI credentials), which flow down through CI/CD hosting layer (GitHub Actions, third-party Action), enter package warehouse Registry(npm registry, PyPI, Docker Hub), and then spread to high-risk npm/PyPI packages (axios, TanStack, litellm, etc.), ultimately impact downstream critical software and clients (web/Node applications, AI/LLM platforms, cloud workloads, enterprise CI/CD, developer hosts).

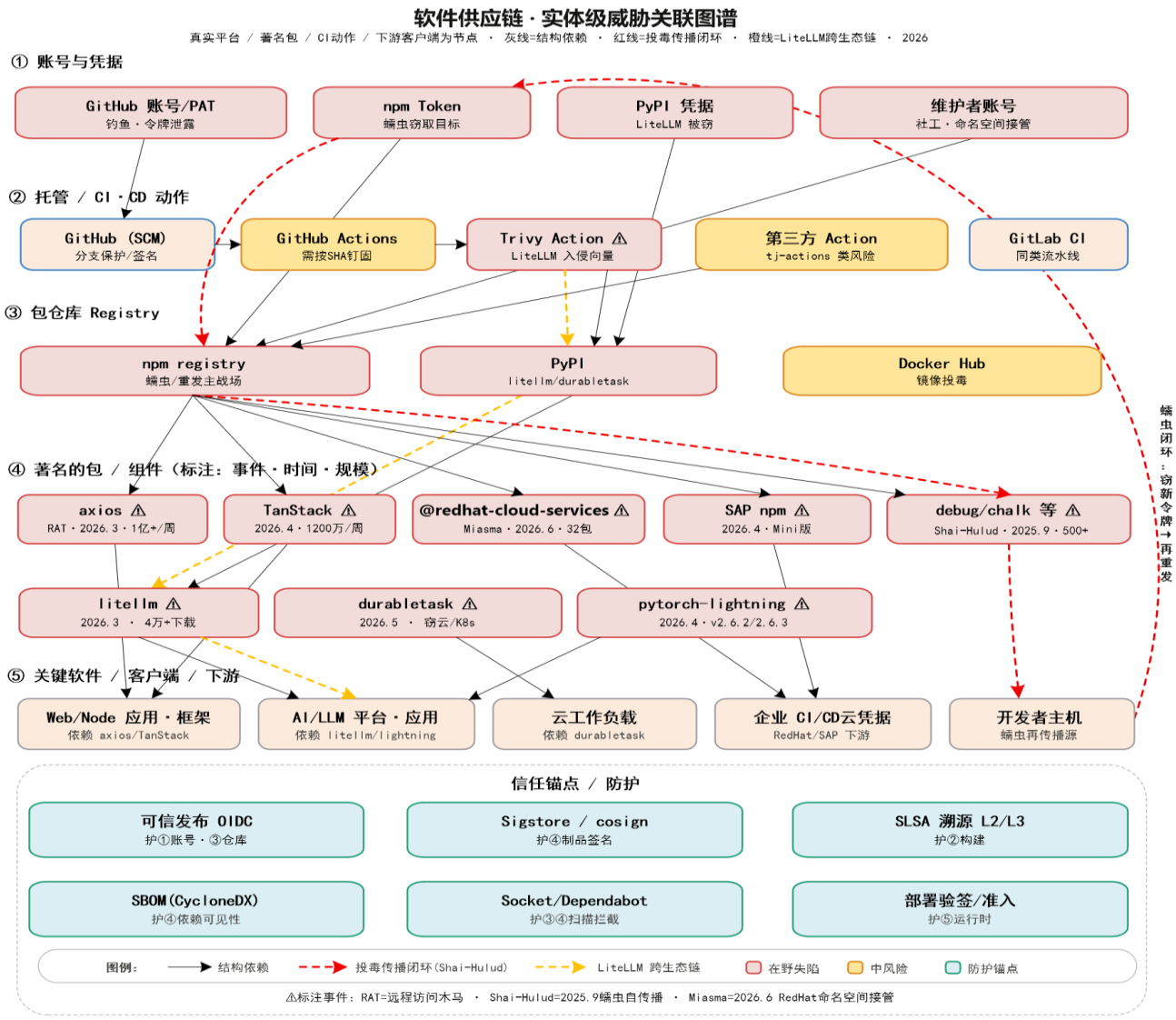


Figure 1-1 Software supply chain and entity-level threat association map1

The graph structure can correspond to each link in the developer tool chain: account credentials (code hosting link) → CI/CD hosting layer (build and release link) → package warehouse Registry (dependency management link) → high-risk open source package (product distribution link) → downstream key software (operation and deployment link). This diagram is a "general diagram" for understanding TeamPCP attack paths, and subsequent chapters will analyze layer by layer along this diagram.

2 Panoramic Map of the Developer tool chain and Risk Mapping

Developers around the world have become accustomed to submitting code on GitHub, introducing dependencies through npm install, clicking on GitHub Actions to trigger builds, and packaging applications into Docker images

and deploying them to the cloud. Each step of this routine relies on a software supply chain of open source tools and third-party services. the core of this attack link is not a simple combination of various components, but a transmission chain of multi-layer trust relationships: when code is pushed to github, the subject default trust platform can ensure account security; When executing npm install dependency installation operation, the default trust package is developed and operated by the regular maintenance party. When triggering the engineering construction process, there is no malicious implantation behavior in the default trust construction environment. Attackers accurately target the security shortcomings of each trust node to implement breakthroughs, without the need to build their own attack carriers, the enterprise's own code warehouse, CI/CD assembly line, package management system, naturally become its optimal intrusion channel.

Figure 2-1 Panoramic map of developer tool chain21

2.1 Code hosting and collaboration platforms are prime targets for source-code theft and credential leaks.

The code hosting environment used in daily development is divided into GitHub, GitLab and self-built hosting Gitea. The team's routine collaboration is completed by means of Pull Request process, code review mechanism and

Issue tracking system. Developers high-frequency use of this type of collaboration features, is the focus of network attackers targeted intrusion entrance. The code hosting platform stores key sensitive assets such as core source code, system access credentials, project configuration files, and personal identity tokens. In the event of security events such as account hijacking and credential disclosure, malicious attackers can tamper with business codes, circumvent the audit and verification process, and implant backdoor programs into programs, causing serious supply chain security risks. The core components of the collaboration platform are disassembled in detail below.

Table 2-1 Code Hosting and Collaboration Platform Core Components List21

Number	Component Type	Typical example
1	Open source code hosting	GitLab/Gitee/Gitea/GitHub (mainstream open source community)
2	Code Review Tool	Gerrit, open source Phabricator, Git native MR/PR mechanism
3	Project & Issue Management	Open source Jira replaces Tuleap, open source Issues component
4	Open Source Documentation Wiki	MkDocs, open source Confluence spin-off project
5	open source code retrieval	OpenGrok, Sourcegraph open source version

2.2 Package management and open-source dependency systems are key targets for supply-chain poisoning

Most modern software projects introduce third-party external code dependencies. when executing dependent pull commands such as npm install, pip install, and go get, it is essentially the same as granting the corresponding tool the operation permission to access the local system, read environment variables, or even execute arbitrary program code. Public warehouses such as npm, PyPI, and Maven Central receive thousands of new packages every day, and their auditing ability is far less than the upload speed. Attackers use this gap to deliver malicious packages in batches. The following are the core risk points for each mainstream language pack management ecosystem.

Table 2-2 List of Mainstream Language Open Source Package Management and Product Warehouse Components22

Number	Component Type	Typical Open Source Example
1	JS package management	npm/yarn/pnpm/deno Open source tools
2	Python Package	pip/poetry/pipenv/uv

	Management	
3	Java Package Management	Maven/Gradle/sbt
4	. NET Package Management	NuGet open source client, Paket
5	Go Module	Go Modules, open source Athens Proxy
6	Rust Package Management	Cargo/crates.io Ecology
7	Open Source Container Image Repository	Harbor open source, GHCR, Docker Hub open source side
8	Private Product Warehouse	Nexus Open Source, Verdaccio
9	System Package Management	apt/yum/dnf/apk/pacman/brew

2.3 Risks of Poisoning in the CI/CD Pipeline Build Phase and Mitigation Strategies

After the code is submitted, GitHub Actions, Jenkins and other tools will automatically complete the whole process of building, testing, product packaging and online deployment. This automated pipeline is the core cornerstone to ensure the efficiency of R & D delivery, and is also the core authority hub that attackers focus on. Once the pipeline configuration is tampered with or the core key in the build environment is compromised, the attacker can automatically implant malicious code in every engineering build process without any sense. TeamPCP attack completed a large-scale software supply chain pollution by hijacking the CI/CD pipeline of TanStack projects. The following section combs the core components and key links that need to be protected in the construction process.

The CI/CD ecosystem covers a fully automated link from code build to deployment. At the managed CI level, GitLab CI and GitHub Actions are the most mainstream choices. They define the build steps by Workflow the configuration, but this also means that once the configuration is tampered with, every commit will automatically perform malicious operations. For teams that need more control, self-hosting solutions such as Jenkins and Drone provide more flexible plug-in extensions, but the plug-in itself may also become a portal for poisoning. In terms of build tools, from traditional Make/CMake to modern Bazel/Vite, the integrity of the build script directly determines the credibility of the output product. Product management builds products through Nexus or GitHub Packages centralized storage, which is a key node in the supply chain. The deployment process involves GitOps tools such as

ArgoCD/Flux and IaC tools such as Terraform/Pulumi, which directly operate the production environment configuration. Once compromised, the scope of influence will expand rapidly.

2.4 Containers and the Kubernetes open-source ecosystem have become extended vectors for supply-chain poisoning in the runtime phase.

The application is packaged as a Docker image and deployed to a Kubernetes cluster, which is often regarded as the end of the delivery process. However, for attackers, the intrusion process has officially started since then. Basic images with backdoors can pollute thousands of downstream business applications in batches. K8s clusters with defective permission configurations provide attackers with access to the global cloud environment. Compared with the construction phase attack, the running state chain poisoning has stronger concealment and greater scope of damage. The core components and corresponding security risks in the container and K8s ecosystem are combed below.

The containerized architecture is the mainstream choice for modern application deployment, and its security ecosystem covers the entire link from image building to runtime management. When the containerd and CRI-O run as the underlying container, they are responsible for the actual execution of the container process. If a vulnerability exists, the attacker may break through the container isolation and directly access the host resources. Kubernetes is the de facto standard for container orchestration, which uses RBAC to control access permissions in the cluster. However, misconfiguration often leads to over-granting of permissions, providing space for attackers to move horizontally. In the K8s ecosystem, Helm is used as a package management tool to simplify application deployment, but the Chart package itself may also carry malicious resources. Policy engines such as OPA/Gatekeeper and Falco are responsible for runtime security detection. If policies are tampered with, security verification will be useless. The image build tool Kaniko/Buildah directly affects the credibility of the underlying image, and once the build context is poisoned, all applications based on the image will be implicated.

2.5 AI/ML open-source development tool chains are a new vector for source-code poisoning.

In the development scenario of code completion with the help of GitHub Copilot, obtaining pre-trained models from the HuggingFace, and building intelligent AI agents based on LangChain, it is easy to ignore a new type of risk: various AI tools have become an emerging breakthrough in software supply chain attacks. An attacker can poison the training data to implant malicious logic into the model from the training phase, embed secret code in the open source

model, or abuse the permission of the AI agent to invoke defects to achieve system penetration. The attack surface is still in the high-speed evolution stage, developers need to establish security awareness as soon as possible. The following section lists the core components and corresponding security risks of AI and ML open source tool chains.

The AI/ML open source tool chain is fast becoming a new addition to the developer toolbox, but its security risks have not been adequately addressed. In terms of code completion assistants, open source kernel projects such as Codeium and various local deployment schemes allow developers to enjoy the convenience of AI-assisted programming, but if the training data is contaminated, the generated code may be "inherently poisonous". Model warehouse HuggingFace has become the preferred distribution platform for pre-trained models. Its open source warehouse and DVC version management tools make model sharing easy, but the downloaded model files may be embedded with stolen code or bundled with malicious dependencies. The training platform Kubeflow directly operate data sets and computing resources. Once it is invaded, all downstream AI applications based on the platform will be contaminated. Agent frameworks such as LangChain and AutoGPT/Dify give AI the ability to call external tools, but the agent's permission overriding and call chain injection may lead to serious security risks. The vector database Chroma/Qdrant/Milvus stores the core retrieval data of the AI application, and the poisoning or data injection of the dependent components may tamper with the retrieval results and steal the project configuration.

2.6 SBOMs, digital signatures, and compliance verification are critical weak points that attackers can exploit to breach trust

The project will introduce a varying number of open source dependent components. The license compliance of the components and the positioning of affected applications after a single open source package exposes malicious code are the core problems in supply chain control. SBOM (Software Bill of Materials) is the core tool to deal with the above problems, which is equivalent to the component ledger of the software project, which can fully trace the source, version information and trusted status of all dependent components. At the same time, the SBOM file itself has the risk of being maliciously tampered with, and the supporting compliance verification process also has the hidden danger of being bypassed by the attacker. The following is a review of the necessary tools for supply chain risk verification and the corresponding security control points.

SBOM is the core infrastructure of supply chain risk verification, covering the complete link from generation to verification. The build tools Syft and Tern can automatically scan project dependencies and output a standardized bill of materials, FOSSology provide more comprehensive license analysis capabilities in the open source compliance

area. But the generated SBOM itself can also be tampered with-if the manifest misses a malicious dependency or hides a risky component, all subsequent security analysis will be based on the error. In terms of parsing validation, CycloneDX and SPDX are the two mainstream standard formats that ensure interoperability between different tools. License Scan Tool ScanCode and FOSSA help identify license conflicts and compliance risks in dependencies. In terms of source trustworthiness verification, Sigstore Rekor provides transparent build attestation records, and SLSA Provenance tools help verify the origin and build process of artifacts-but as TeamPCP attacks show, these trust attestation themselves can be forged.

2.7 Both third-party services and trust boundaries have become attack entry points.

The security risks of R & D core tool chains, such as code hosting, dependency management, CI/CD, container running, and AI tools, were discussed earlier. However, modern application architectures are highly open and rarely operate in isolation: business systems generally interface with external capabilities such as third-party payments, external identity authentication, public SaaS services, and partner APIs. Various third-party integration and external interconnection capabilities, while greatly improving the efficiency of research and development and business landing, continue to broaden the software supply chain attack surface. The intruded partner SaaS service can be used as a springboard for horizontal penetration. Access to a third-party SDK with a back door can directly implant malicious logic into the business system. Over-authorized OAuth permissions will also cause permission overflow, leading to the risk of unauthorized access. Such external dependencies are often not fully controlled by their own parties, but their own security shortcomings can be directly transmitted to the local business system, causing systemic security risks. The following section sorts out the core components and key links that need to be protected in the external connection scenario.

The security risks of third-party integration scenarios are mainly concentrated in multiple core dimensions. If the identity authentication integration system (OAuth, SAML, OIDC, etc.) has configuration defects, attackers can abuse the cross-domain trust relationship to achieve unauthorized access. Third-party SDKs and open-source components introduced in scenarios such as data analysis, message push, and social sharing can easily be embedded with malicious code and latent backdoors if the source is not trusted. Once the partner API interface is breached, the attack payload can be backpenetrated to your business system through the interface response data. If the Webhook mechanism does not deploy a strict signature verification policy, attackers can forge malicious callback requests and illegally trigger internal business operations. In addition, if the operation and maintenance monitoring platform such

as log and alarm is invaded, it will not only cause the core sensitive data to be leaked, but also the attacker can rely on the inherent authority of the platform to implement deeper vertical and horizontal penetration. The following are typical risk scenarios that need to be vigilant in external connection scenarios.

The core difficulty of software supply chain security is that the security status of various external dependencies cannot be directly controlled by one's own party, but the security vulnerabilities and defects that exist can directly affect one's own business systems. Therefore, the establishment of a sound access audit, continuous monitoring and emergency response system for third-party services is as critical as its own code security reinforcement and internal authority control, and is the core link of building a solid supply chain security defense line.

Sorting out the risks of various supply chain tools and components is only the basic level of security awareness, and more importantly, it is important to understand the attacker's use logic: attackers can rely on various R&D and third-party tools that enterprises rely on daily to transform normal development and operation operations into available intrusion channels. The following section will switch to the attacker's perspective and disassemble the fall and utilization process of various mainstream tools in real attack and defense scenarios in combination with TeamPCP real attack actual combat cases. Different from simply listing risk items, the core of this section is to analyze in depth the complete attack link of conventional research and development tools being broken through and abused, and clearly explain the collapse principle of the existing business dependence system.

3 Systematic inventory and analysis of the attack surface

The previous chapter draws a panoramic map of the tool chain from the daily perspective of developers-code hosting, package management, CI/CD, container operation, AI tools, compliance verification, external integration, a total of 7 links. Each link is a pillar of development efficiency and a channel for attackers.

Antiy CERT is carried out one by one according to the same workflow, but the perspective shifts from "how developers use" to "how attackers break". Each link follows the structure of "scenario introduction → attack logic analysis → key vector extraction → developer's response ideas". sorting out the risks of various supply chain tools and components is only the basic level of security cognition. more importantly, it is to understand the attacker's utilization logic: attackers can rely on various R & D and third-party tools that enterprises rely on daily to transform normal development, operation and maintenance operations into usable intrusion channels. This article will take TeamPCP real attack and defense cases as the core entry point, one by one to disassemble the abuse methods and fall

process of each tool link in the real attack scenario. The content abandons the broad and idealized list of risks and focuses on core security risks that have been proven in the field, directly affect the development process, and pose a substantial threat to business systems. The full text of the final supporting comprehensive risk matrix, can intuitively quantify the risk level of each link, for the follow-up security reinforcement, hidden danger rectification and protection strategy optimization to provide a clear priority basis.

The following table shows the mapping between the tool chain link and the attack surface, which helps readers quickly associate the attack scenarios in this chapter with the front basic content of the tool chain panorama in Chapter 2 to form a complete knowledge loop.

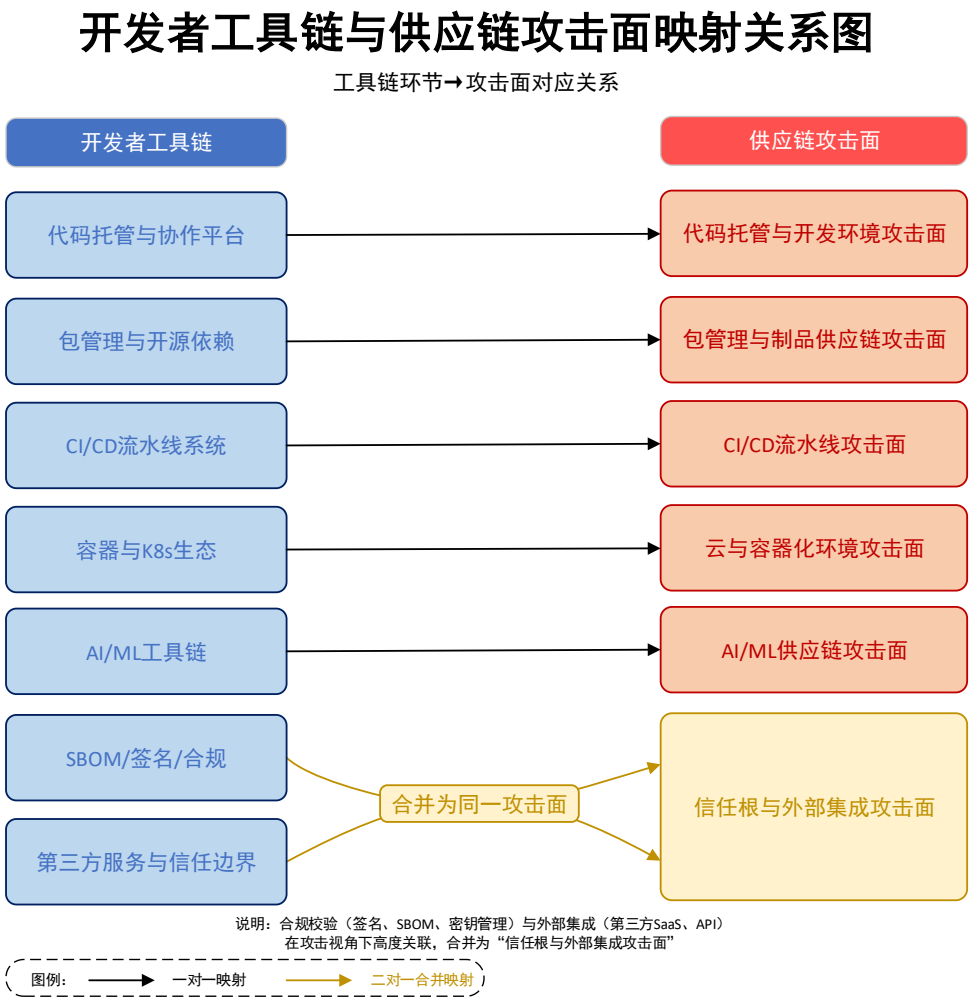


Figure 3-1 Mapping diagram of developer tool chain and attack surface31

Each attack surface follows the structure of "Scenario Introduction → Attack Logic Analysis → Key Vector Refinement → Developer Response Ideas", starting from TeamPCP actual combat cases and focusing on the key risks that have occurred in actual combat.

3.1 Attack Surface of Code Hosting and Development Environments

The code hosting platform and the developer's local environment are the "source" of the entire supply chain-this link retains core assets such as source code, access credentials, project configurations, and personal tokens. Once the GitHub account is hijacked, the attacker can directly modify the code, bypass the review, and implant the backdoor. Once the developer workstation is breached, the SSH private key, Git credential, and cloud access token stored locally will all be exposed. TeamPCP's Rope and Koschei tools are specifically designed to traverse developers' local file systems to collect sensitive credentials, while BreachForums reward mechanisms induce developers to actively execute malicious programs. The attacker's methods on the source side include account hijacking and credential theft, malicious code submission and branch tampering, development tool poisoning and IDE plug-in backdoors, local key search and configuration file disclosure. Here are the key attack vectors distilled from TeamPCP combat.

Table 3-1 Code Hosting and Development Environment Key Attack Vectors31

Number	Attack vector	Risk attribute	Attack method	TeamPCP Mapping
1	Warehouse account hijacking	Extremely high	Stealing Maintainer Credentials to Push Malicious Updates	Chalk/Debug Update Pollution
2	Malicious Code Submission	High	Bypass code review, implant backdoors, tamper with branch protection rules	Megalodon action
3	Developer Token Leak	Extremely high	PAT/npm Token is exposed in plaintext and stolen by attackers in batches.	Shai-Hulud worm propagation
4	Local Key Search	High	Traversing the developer's local file system to steal SSH keys and cloud credentials	Rope/Koschei Local Collection
5	IDE plug-in backdoor	middle	Implant malicious logic as a development tool extension	Common Entry Points for Supply Chain Attacks
6	Development Tools Poisoned	High	Tamper compiler, build tool installation package	Common Entry Points for Supply Chain Attacks
7	Social worker induced attack	middle	Reward Competition Induces Developers to Execute Malicious Programs	BreachForums reward
8	Profile Disclosure	middle	The. env and config files were mistakenly submitted to the public repository.	Common Entry Points for Supply Chain Attacks

3.2 Package management and the attack surface of the artifact supply chain

Every dependency package a developer introduces through the npm install is essentially a program written by a stranger and distributed through a third-party platform. When these programs contain malicious logic, they are executed at the moment when you are least protected-the installation phase. TeamPCP Shai-Hulud worms take advantage of this timing: through postinstall hooks, when users install seemingly normal npm packages, they automatically steal local credentials and spread them out. Attackers use much more than this in the dependency chain-from publishing counterfeit package names to hijacking discarded high-download packages, from polluting deep dependencies to tampering with product releases, the following are key attack vectors extracted from TeamPCP combat.

Table 3-2 Key Attack Vectors for Package Management and Product Supply Chain32

Number	Attack vector	Risk attribute	Attack method	TeamPCP Mapping
1	Malicious package upload	High	Upload counterfeit malicious packages with the same name in a public repository	React2Shell, Mini Shai-Hulud
2	Legitimate package hijacking	High	Stealing maintainer accounts to push malicious updates	Chalk/Debug update pollution
3	Dependency obfuscation attack	middle	A public malicious package with the same name as an internal package.	Common Entry Points for Supply Chain Attacks
4	Name confusion poisoning	middle	Register Spelling Approximate Fishing Package Name	Common Entry Points for Supply Chain Attacks
5	Install hook execution	Extremely high	Automate malicious code execution with lifecycle scripts	Shai-Hulud Worm
6	Abandoned Package Takeover	middle	Apply to take over long-term unmaintained high-download packages	Common Entry Points for Supply Chain Attacks
7	deep dependence pollution	Extremely high	Pollution Mass Projects Share Bottom Dependency Library	Chalk Component Poisone
8	Version Range Hijacking	middle	Loose version rules to implant malicious patches	Common Entry Points for

				Supply Chain Attacks
9	binary artifact replacement	High	Replacing regular binary files in the release phase	Common Entry Points for Supply Chain Attacks
10	SBOM Information Manipulation	High	Tampering with Manifest to Hide Malicious Dependent Components	Common Entry Points for Supply Chain Attacks

3.3 CI/CD Pipeline Attack Surface

For attackers, breaching the CI/CD pipeline enables continuous automated intrusion. Attackers do not need to poison the code warehouse one by one, only need to tamper with the pipeline configuration once, can automatically trigger malicious execution logic in each subsequent code submission and engineering construction process, to achieve batch, persistent supply chain pollution. TeamPCP hijacked TanStack's official CI/CD pipeline in the Mini Shai-Hulud attack, stole OIDC identity tokens, and even successfully forged trusted publishing credentials that meet SLSA L3 standards. This means that the malicious program has obtained the same credible identification as the official genuine product. The following are the key attack vectors for the CI/CD pipeline.

Table 3-3 CI/CD Pipeline Key Attack Vectors33

Number	Attack vector	Risk attribute	Attack mode	TeamPCP mapping
1	CI/CD Configuration Injection	middle	Implant malicious steps through PR modification workflow	Common Access to Supply Chain Attacks
2	OIDC Token Stealing Abuse	Extremely high	Stealing pipeline trusted identity credentials	TanStack CI/CD Hijacking
3	Unexpected key leakage	middle	Key output to log or packaged into product	Common Entry Points for Supply Chain Attacks
4	Runner Environmental Pollution	High	Self-hosted node implantation of persistent malicious programs	Common Entry Points for Supply Chain Attacks
5	Third-party component hijacking	middle	Introduce the compromised Action, plug-in	Common Entry Points for Supply Chain Attacks
6	Build Script Tampering	middle	Modify malicious logic embedded in compiled scripts	Common Entry Points for Supply Chain Attacks

7	Build Cache Poison	High	Contaminate npm, container build cache	Common Entry Points for Supply Chain Attacks
8	Proof of falsified source	Extremely high	Forged SLSA Compliance Issued Voucher	Breakthrough SLSA L3
9	Deployment Permission Abuse	middle	Deliver malicious code with pipeline permissions	Common Entry Points for Supply Chain Attacks
10	Pipeline configuration tampering	High	Roundabout Change Pipeline Configuration	Common Entry Points for Supply Chain Attacks

3.4 Attack Surface of Cloud and Containerized Environments

When a business application is deployed on public cloud platforms such as AWS, Alibaba Cloud, and GCP, the system is granted access to cloud resources. The authorization carrier includes API keys, service accounts, and temporary access tokens. Compared with business data, such identity credentials have higher attack value. Once the AK/SK of the cloud primary account is leaked, the attacker can obtain complete control permissions for the entire cloud environment. The TeamPCP attack link lists cloud credentials as the primary target of theft, relying on the stolen credentials to implement horizontal penetration, long-term residence, and then invade the rest of the enterprise business system. The following section combs the core attack vectors of the cloud infrastructure dimension.

Table 3-4 Key Attack Vectors for Cloud Environments34

Number	Attack vector	Risk Attributes	Attack method	TeamPCP Mapping
1	Cloud Key Stealing Abuse	Extremely high	Stealing AK/SK and calling the cloud interface	Rope/Koschei credential theft
2	IMDS Interface Abuse	High	Relying on the old metadata interface to steal temporary credentials	Common Entry Points for Supply Chain Attacks
3	Cloud Configuration Exploit	Low	Overreach with open storage, loose security group access	Common Entry Points for Supply Chain Attacks
4	Service account key theft	High	Stealing Cloud Principal Key Horizontal Penetration	Rope/Koschei
5	Cloud function code injection	middle	Event Trigger Implantation of Malicious Code Execution	Common Entry Points for Supply Chain Attacks
6	API Key Code Disclosure	Low	Keys hard-coded exposed in open source code	Common Entry Points for Supply Chain Attacks

7	cross-tenant ultra vires attack	High	Using sharing mechanism to realize cross-account rights raising	Common Entry Points for Supply Chain Attacks
8	Poisoned Cloud Development Environment	High	Cloud IDE buries persistent malicious configuration	Common Entry Points for Supply Chain Attacks

The containerized architecture greatly simplifies the deployment and elastic scaling process of applications, but at the same time, it gives rise to a new supply chain attack path. The tampered Docker basic image can insert latent backdoors into business systems without awareness. Configuring an irregular Kubernetes RBAC permission policy may lead to the leakage of cluster permissions and enable attackers to obtain the control permission of the entire cluster. A malicious Helm Chart can silently create a privileged container during the resource deployment phase and reserve an intrusion portal. Compared with conventional attack methods, container environment risks have typical chain diffusion characteristics. After a single-point container falls, the attacker can use this as a fulcrum to complete horizontal and vertical penetration of the host, other containers in the same cluster, and even the global cloud network layer by layer. The core attack vectors in the container and K8s ecosystem are disassembled in detail below.

Table 3-5 Key Attack Vectors for Containerized Environments35

Number	Attack vector	Risk attribute	Attack method	TeamPCP Mapping
1	Container Mirroring Exploit	Low	Use the known CVE vulnerability of the image to obtain container permissions and further break the isolation.	Common Risks of Open Source Ecosystem
2	Container Escape Attack	High	Exceeds container isolation through privileged container configuration errors or runtime vulnerabilities to gain control of the host.	Koschei lateral movement technique
3	K8s API unauthorized access	Low	The K8s API server exposed on the public network is directly called, and the attacker obtains the list of all cluster resources.	Common Entry Points for Supply Chain Attacks
4	Horizontal lifting of RBAC authority	middle	Use over-authorized service accounts to move horizontally within the cluster to gradually expand the scope of control.	Common Entry Points for Supply Chain Attacks
5	Cluster Secret	middle	Extract sensitive credentials such as	Rope/Koschei

	Stealing		database password and API key from K8s Secret or ConfigMap	voucher collection
6	Helm Chart bag contamination	middle	Malicious resource definitions are built in the chart package, and backdoor services or privileged containers are automatically created during deployment.	Common Entry Points for Supply Chain Attacks
7	Access Policy Bypass	High	Tamper or bypass OPA/Gatekeeper admission control policies to deploy unverified malicious payload	Common Entry Points for Supply Chain Attacks
8	mirror warehouse replacement attack	middle	After the mirror warehouse is captured, replace the online mirror label and distribute the malicious version to all pulpers.	Common Entry Points for Supply Chain Attacks
9	Sidecar/Init container poisoning	High	A backdoor is implanted in the Sidecar or initialization container and automatically activated with the main application deployment	Common Entry Points for Supply Chain Attacks
10	Network policy lateral movement	High	Use K8s network policy vulnerabilities or DNS hijacking to move laterally between pods to expand the attack surface	Koschei Cloud Infiltration Technique

3.5 AI/ML supply chain attack surface

AI technology is reshaping the software development model at a high speed, while giving birth to a number of new security attack surfaces. When using tools such as Copilot to generate code, malicious logic may be embedded in the code completion content output by the tools. When obtaining a public pre-trained model from a platform such as HuggingFace, the authenticity of the source of the model itself and the integrity of the file lack reliable verification means. After the external API calling permission is opened for the AI Agent, if its operation boundary is not strictly restricted, it is easy to cause the risk of ultra vires. At present, a mature and standardized defense system has not yet been formed for such AI native risks. Attackers have regarded the AI/ML supply chain as a high-threat attack blue ocean and continue to exploit it. The core attack vectors in the AI and ML supply chain are reviewed below.

Table 3-6 AI/ML Supply Chain Key Attack Vectors36

Number	Attack vector	Attack method	Current Risk	Trend

1	Poisoned training data set	Contaminated sample tampering model output logic	Low	↑ Up
2	AI generated code leakage review	Large model output code hidden vulnerabilities	Low	↑ Rise
3	model stealing extraction	interface query reverse restore model	middle	→ Smooth
4	Combatting Sample Interference	Special input misleading model judgment	middle	→Stable
5	Prompt injection	Construction Special Command Breaks AI Limit	middle	↑ Rise
6	Vector Library Injection	Malicious vector tampering retrieval results	Low	↑ Rise
7	Open source model with poison	Open source platform download embedded backdoor model	Low	↑ Rise
8	AI Agent Permissions Out of Control	Agent Super Privilege Call System Capability	Low	↑ Rise
9	Sensitive Code Outposted	Source code paste AI cause information leakage	middle	→Stable

3.6 Trust root and external integration attack surface

The previous analysis focuses on R & D tool links that developers use and operate directly, covering core links such as code hosting, open source dependencies, CI/CD build, business deployment, and AI development tools. However, under this whole link, there is an invisible supporting system running silently: identity authentication manages who has access to what, key system protects the credibility of communication and signature, third-party SDK and SaaS services expand the functional boundary, API gateway controls the data flow of internal and external systems. These links do not directly generate code, but they define the "trust boundaries" of the entire supply chain ". Once the identity system is bypassed, the key is leaked, the third party is breached, and the API is abused, all previous security investments will be wasted. The following key attack vectors are presented by four sub-topics-from identity and access management, to data and key management, to third-party and SaaS integration, and finally to API and web security.

1. Identity and Access Management -- Account Credentials, Authentication Mechanism and Permission Control

Table 3-7 List of Identity and Access Management Attack Vectors37

Number	Attack vector	Risk	Attack method
1	account password leakage cracking	Low	Hit the database and brute force crack to obtain the account number
2	MFA Authentication Bypass	middle	Social worker means to bypass secondary identity verification
3	Service account key theft	Extremely high	Batch search of various types of service credentials
4	Session Hijacking	middle	Stealing cookies to use legitimate sessions
5	IAM Permission Emissions	High	Expanding Access Permissions by Relying on Misconfiguration
6	unified authentication platform intrusion	High	Attack IdP and hijack full account trust
7	Key public network leakage	Low	Credentials submitted to open source repository by mistake
8	Resignation Permission Remaining	middle	System Permission Not Recovered for Personnel Change
9	Illegal API key call	middle	The leaked interface key is maliciously exploited.
10	Cross-domain trust abuse	High	Abuse of OIDC/SAML trust link penetration

2. Data and Key Management -- Database Connection, Key Storage and Encryption System

Table 3-8 List of Data and Key Management Attack Vectors3-8

Serial Number	attack vector	Risk	Attack method
1	Database connection information disclosure	Low	Database intrusion caused by configuration plaintext leakage
2	Key Management Platform Break	High	Intrusion KMS Batch Steal Encryption Key
3	Transmission traffic eavesdropping	middle	Unencrypted link intercepts business sensitive data
4	Decryption of stock data	High	Key leakage to crack encrypted business data
5	Signature certificate private key theft	middle	Steal certificates and forge legitimate signatures
6	Profile disclosure	middle	Configure the password of the system leaked account
7	Backup file disclosure	middle	Unencrypted backup causes data breach
8	IaC configuration leak	middle	Terraform configuration plaintext retention key

3. Third-party integration with SaaS-OAuth authorization, SDK dependency, and external service access

Table 3-9 List of Third-Party and SaaS Integration Attack Vectors39

Number	Attack vector	Risk	Attack method
1	Third-party interface key disclosure	Low	Abuse Caused by Hard-Coded Key Disclosure
2	OAuth authorization abuse	middle	Authority transboundary caused by unreasonable authorization
3	Webhook request forgery	middle	Forged third-party callbacks to trigger internal operations
4	Collaboration SaaS Supply Chain Fall	High	Upstream services are poisoned and implicated in their own business.
5	Third-party SDK embedded backdoor	middle	Introducing development components with malicious logic
6	Cooperative Link Hijacking	middle	Intercept partner data transmission content
7	Operation and maintenance platform intrusion	High	Trapping Log Monitoring System to Steal Information
8	CI third-party component contamination	middle	The pipeline plug-in and template have been maliciously tampered.

4. API and Web Security Layer-Interface Authentication, WAF Protection and CDN Security

Table 3-10 API and Web Security Layer Attack Vector List310

Number	Attack vector	Risk	Attack method
1	Interface unauthorized access	Low	The API lacks authentication and is called arbitrarily.
2	Excessive return of interface data	Low	Interface disclosure of information beyond business requirements
3	Various types of injection attacks	middle	Parameter Splicing Trigger Injection Vulnerability
4	throttling policy bypass	middle	Violent scan through restrictions
5	WAF Rule Bypass	High	Code Deformation Bypass Protection Policy
6	CDN cache poisoning	High	Tampering cache to deliver malicious resources
7	DNS resolution hijacking	middle	Tampering with domain names pointing to malicious sites
8	Traffic-based DDoS	Low	Massive Requests Exhausting Business Resources
9	Third-party API key	middle	Cooperative interface key leakage and abuse

	disclosure		
10	Old Obsolete Interface Vulnerability	middle	Uncleaned Vulnerabilities Left in the Downline Interface

3.7 Comprehensive Attack Surface Risk Matrix

The core attack vectors for the six attack faces have been fully disassembled and analyzed. Combined with the reality of enterprise security budget constraints, the following focus on the direction of resource priority to give landing guidance. The table measures the comprehensive risk level of each module around the four evaluation dimensions of attack frequency, risk impact range, safety detection difficulty and fault repair cost, and marks the highest priority disposal measures, which can be directly used for the scheduling and resource allocation of safety rectification work.

Table 31-1 Multi-dimensional supply chain attack surface risk comprehensive assessment matrix

Attack surface	Attack frequency	Scope of impact	Detection difficulty	Repair cost	Highest priority measures
Code hosting and development environment	Extremely high	Extremely wide	middle	middle	Branch Protection • MFA • Terminal Security Control • Key Pre-detection
Package Management and Product Supply Chain	Extremely high	Extremely wide	Low	Low	Private Source Forced Routing, Dependency Lock (SHA Fixed), SCA Scan
CI/CD Pipeline	High	Extremely wide	High	High	Workflow Review • OIDC Least Privilege • SLSA Traceability
Cloud and containerized environments	High	wide	High	High	Voucher Centralized Management • IMDS v2 • Admission Control • Mirror Signature
AI/ML Supply Chain	Low (rising)	Extremely wide	Extremely high	Extremely high	AI Usage Specification • Generate Code Review • Model Source Validation
Root of Trust and External Integration	Extremely high	Extremely wide	High	middle	[Identity] Full MFA [Key] Platform Escrow [Third Party] Access Review [API] Full Authentication

Based on the developer workflow, the above systematically combs the six types of attack surfaces, and restores the complete intrusion links of TeamPCP attack events, and analyzes the collapse points of various risks in the supply

chain system. However, the scattered attack vectors are only surface risk characteristics, and the core that really drives the supply chain security crisis is the structural factors such as the rapid expansion of the AI ecology and the increasing complexity of the system, which are coupled to each other to form a threat flywheel that continues to iterate and accelerate evolution. To build an efficient and sustainable defense system, we must not only limit ourselves to single-point vulnerability repair, but also have deep control over the underlying structural risk drivers.

The construction of security protection needs to complete the upgrade from "identifying single-point risks and clarifying the point of collapse" to "global system governance and systematic defense. Based on the full life cycle of software delivery, this paper builds an eight-link trust management and control framework, disassembling the core trust nodes of each stage layer by layer, and clearly corresponding attack links and landing protection schemes. The framework and the previous attack surface analysis form complementary support: attack surface analysis accurately locates high-risk risk points, and the trust chain control system standardizes the definition of the whole process protection action, realizing the closed loop from risk perception to landing governance.

Table 3-12 Software Supply Chain Lifecycle Trust Chain Control and Protection Scheme3

Chain of Trust	Attack Surface (Poison Method)	Prevention and reinforcement
Developer identity and credentials	**Account/token theft • phishing • social worker takeover • weak password → credential is "master key 」**	**Mandatory MFA • Hardware Security Key (FIDO2) • Short-time Token • Minimum Permissions • Credential Rotation**
Source Code and Version Control	**Malicious submission • Branch/label tampering • Warehouse backdoor • Hidden Unicode characters**	**Submit Signature (GPG/Sigstore) • Branch Protection • AI Auxiliary Code Pre-Trial • Double Code Review • Protected Label**
Open Source Dependencies and Components	**Poison-cybersquatting (typosquat)-dependency obfuscation-malicious maintainer-self-propagating worm**	**Locked Version According to Summary (SHA) Fixed SCA (Software Component Analysis) Scan, Dependent Whitelist, Generate SBOM.**
Build vs. CI/CD	**Build pollution, third-party Action poisoning, pipeline key leakage, persistent residence.**	Isolation/Reproduction Build • SLSA Traceability • Action Press SHA Fixing • Minimum Permission
ARTICLES AND SIGNATURES	**Products are replaced/tampered with, unsigned distribution,**	Sigstore/cosign signature, build proof hash check, tamper-proof release certificate.

	middleman injection, hash is not verified.	
Distribution and Warehouse	**Warehouse Account Loss, Malicious Version Reissue, Namespace Hijacking, Mirror Pollution**	**Warehouse 2FA • Trusted Release (OIDC Token Free) • Immutable Release • Release Approval**
Deployment and operation	**Pull variable label (latest), run injection, go online without verification, and have too much permission.**	**Access control, mandatory check-in at the time of deployment, according to the summary of the fixed, run-time minimum permissions.**
Monitoring and Response	**Detection lag • Lack of SBOM visibility • No contingency plan • Unknown impact scope**	**Continuous Monitoring • SBOM Retrospective Investigation • Threat Intelligence • Emergency Plan and Exercise**

This table combs the R & D tool chain control system under the full link trust dimension, and the risks of the eight trust links have strong chain conduction characteristics. The loss of upstream identity and source code will cause global cascade pollution, and downstream deployment and monitoring short boards will continue to amplify the losses caused by attacks. Global protection relies on multi-layer control mechanisms to coordinate landing, identity credential control, source code signature verification, reliance on fingerprint constraints, pipeline authority convergence, product trusted signature verification, operation access interception, intelligent monitoring response and other measures are deployed layer by layer, multi-dimensional constraints overlay to form a complete supply chain defense-in-depth structure.

4 A Deep Deconstruction of Supply Chain Threats

The complexity of the tool supply chain system is superimposed on the TeamPCP "sandstorm" poisoning model, resulting in an extremely complex overall risk transmission path. We distill the nine core elements and variables that are evolving behind it: ecological generalization, explosion of exposure, complexity expansion, attack technology leaps, value center of gravity migration, trust system collapse, defense responsibility mismatch, and governance detection lag. To better analyze the hierarchical effects of these elements and variables over time and in the environment, we map them to six functional levels: the scenario layer, the expansion layer, the threat layer, the value layer, the effect layer, and the defense layer. These nine elements and variables do not exist independently and in parallel, but form a six-layer causal transmission chain-from the scene layer to the underlying defense layer, where

the deterioration of each layer directly or indirectly drives the loss of control of the next layer, ultimately forming a self-reinforcing positive feedback closed loop.

- Scene layer: composed of ecological generalization and exposure surge, is the engine of the threat chain.
- Expansion layer: Complexity expands so that potential threats are exponentially amplified.
- Threat layer: The attack technology after the transition converts the above potential threats into actual attack capabilities.
- Value layer: The migration of the value center determines the attacker's target priority.
- Effect layer: The collapse of the trust system and the expansion of the attack economy trigger a chain reaction at the system level.
- Defensive layer: defensive responsibility mismatch and governance detection lag is the inevitable result of the imbalance of the pre-order dimension, but also the starting point of a new cycle.

A thorough understanding of this causal chain is a cognitive prerequisite for developing effective defense strategies.

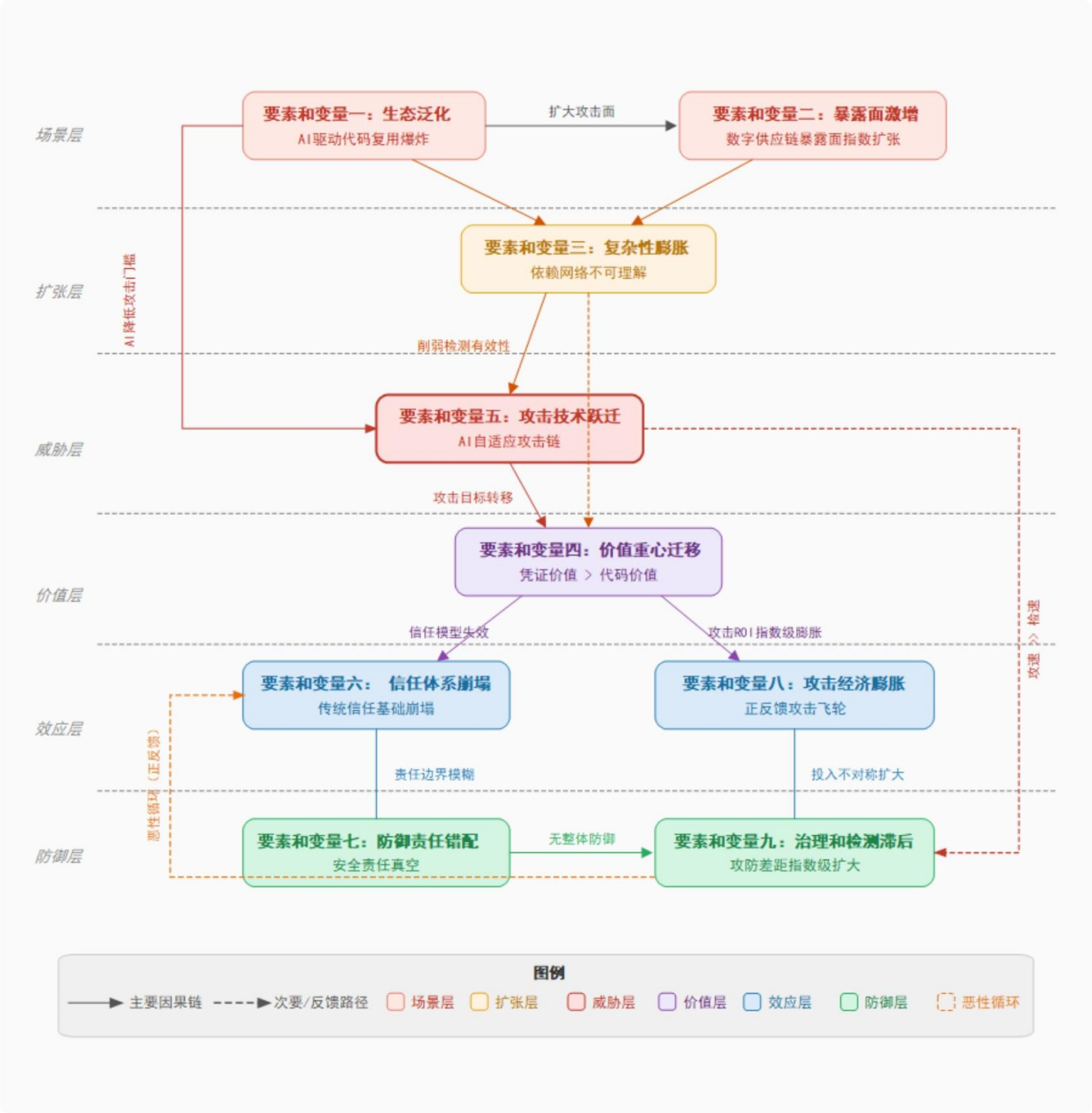


Figure 4-1 Multi-factor and variable supply chain threat multi-layer causal architecture1

Six-layer causal architecture description: The above figure presents the causal transmission between multiple elements and variables in a hierarchical architecture. The solid arrow represents the primary causal chain, which is conducted from the scene layer to the defense layer from top to bottom; the dashed arrow represents the secondary feedback path, which constitutes three positive feedback coupling loops. The "scene layer", "expansion layer", "threat layer", "value layer", "effect layer" and "defense layer" marked on the left side identify the functional positioning of each element and variable in the attack link: the scene layer provides the original power, the expansion layer enlarges

the scope of influence, the threat layer transforms the threat into the actual attack, the value layer redefines the attack target, and the effect layer produces systematic consequences, the defense layer exposes governance gaps and triggers a new cycle.

4.1 Scenario Layer: Ecological Generalization and a Surge in the Attack Surface Create a Risky Environment

The scene layer consists of element and variable one (ecological generalization) and element and variable two (exposure surge), which is the original source of power for the entire multi-factor and variable causal chain. The popularity of AI technology has fundamentally changed the ecological structure of software development, and this ecological generalization has directly led to the exponential expansion of attack exposure. The first is the "cause" and the second is the "effect", which together constitute the first driving force of risk generation and diffusion.

Elements and variables one: ecological generalization-the structural shift from "centralized distribution" to "decentralized collaboration. The software supply chain ecosystem is undergoing a profound restructuring accelerated by AI. The average daily code generation of AI programming tools such as GitHub Copilot and ChatGPT has exceeded the total amount of manual writing by human developers worldwide, meaning that AI are spawning new dependencies at an alarming rate. Each AI-generated code snippet can inadvertently introduce new library calls, new framework dependencies, or new API integrations that would have been subject to rigorous manual review and security assessment under the traditional model. However, in AI-assisted development environments, this review is often omitted or simplified-developers tend to blindly trust AI-generated code and ignore the link risks behind it.

The prosperity of the open source ecosystem further amplifies this trend. The number of packages in public warehouses such as npm, PyPI and Maven Central has exceeded 5 million, with thousands of new packages added every day. The intricate network of dependencies between these packages is being continuously injected with new nodes and edges by AI code. TeamPCP's Shai-Hulud worm exploits this ecological vulnerability-it masquerades as a legitimate npm package and looks for a propagation path through a huge network of dependencies. When the ecological scale exceeds the ability of human intuitive management, the attacker finds a breakthrough in landing external threats.

Element and variable two: Exposure surge-from "code package" to "full development link". The direct consequence of ecological generalization is the dramatic expansion of exposed surfaces. In traditional supply chain attacks, the attacker's target is mainly focused on the end user's equipment and data; in the AI era, the target of the

attack has fundamentally shifted. Every link in the full development link becomes a potential attack portal, including developer workstations, CI/CD pipelines, cloud infrastructure, container image repositories, API gateways, and so on. TeamPCP attack practices clearly demonstrate this kind of exposure expansion: from stealing GitHub developer accounts (identity layer), hijacking npm to publish Token (package management layer), polluting PyPI packages (distribution layer), using infected dependency packages to move horizontally in CI/CD pipelines (build layer), and finally deploying persistent backdoors (run layer) in enterprise cloud environments. The attack path is no longer limited to a single link, but runs through the entire link from code writing to production deployment.

The causal transmission logic of ecological generalization is clear and dangerous: AI-driven code reuse explosion → ecological scale expansion out of control → exposed surface exponential expansion. This is a self-accelerating vicious cycle-AI generate more code → code introduces more dependencies → dependencies create more exposed surfaces → larger exposed surfaces attract more attackers → attackers use AI to generate more complex attack code → loop acceleration. The reason why TeamPCP Typosquatting attacks and dependency injection attacks can affect thousands of projects in a short period of time is precisely because they accurately capture and take advantage of the "attack dividend" brought by this exposure expansion ".

4.2 Expansion Layer: Complexity Inflation Gives Rise to the Dilemma of Network Dependency and Governance

The expansion layer is composed of three elements and variables (complexity expansion) and is the amplifier of the downward conduction of the scene layer. Ecological generalization and the proliferation of exposed areas directly lead to the complexity explosion of dependent networks, which in turn weakens the effectiveness of security detection and provides more room for attackers to operate. The element and variable three connects the scene layer and the threat layer-it is not a threat in itself, but it makes the threat more difficult to find, more difficult to stop, and becomes a "fog".

Elements and variables III: complexity expansion-the untraceability of the supply chain from a "chain" to a "network.

The dependencies of modern software projects go far beyond the "chain" metaphor-they are more like an intricate web:

- There are many nodes: each package may be connected to dozens or even hundreds of other nodes, and these connections are nested layer by layer, forming a recursive structure with a depth of up to dozens of layers.

- The scale is staggering: a typical enterprise project may rely on 50-200 packages directly, while the total number of indirect dependencies may reach 10000-100000, and the depth of the dependency tree can reach 15-30 layers.

- Artificial Disability: The sheer size of this network is beyond the capacity of any human review.

One dimension of complexity is the diversity of dependent sources. The same project may use packages from multiple sources such as npm, PyPI, Maven Central, Docker Hub, GitHub Packages, private repositories, etc., each with its own security policy, signing mechanism, publishing process, and trust model. This heterogeneity makes it extremely difficult to unify security policies-security measures that are effective at one source may not exist at all at another source. TeamPCP Mini Shai-Hulud attacks take advantage of this heterogeneity: malicious payloads are delivered through PyPI packages in the Python ecosystem, and then the cross-language call mechanism between Python and JavaScript is used to spread the attack to the npm ecosystem, successfully bypassing the security detection of a single ecosystem.

The second dimension of complexity is temporal dynamics. The dependency network is not static-package versions are constantly being updated, new dependencies are being added, and old dependencies are being deprecated. This dynamic nature makes the "one-time security review" unable to take effect for a long time, and a continuous security monitoring mechanism must be established, which further increases the complexity of security operations. TeamPCP persistent worms take advantage of this dynamism-the backdoors they implant in a system can be activated weeks or even months later, when the initial attack traces are already covered by daily system updates and log rotations.

The causal transmission logic of complexity expansion is: continuous expansion of ecological scale → exponential expansion of network complexity → failure of detection tools and analysis methods → greater concealment space for attackers → increased attack success rate → more attackers entering the field → further expansion of ecology (positive feedback).

4.3 Threat Layer: AI constructs adaptive payload attacks that progressively penetrate the trust system layer by layer.

The threat layer consists of five elements and variables (attack technology transitions) and is the key hub for translating the impact of the scenario and expansion layers into actual attack capabilities. The speed at which defenders embrace AI lags far behind the escalation of attack means-the asymmetry of this speed leads to a generational jump in attack technology far faster than defense technology follows. Element and variable five connect the scene/expansion layer with the value layer-it transforms "more exposed" and "deeper concealment" into "more accurate and efficient" attack capabilities.

Element and variable five: Attack technology transition-AI-enabled adaptive attack chain. The evolution of TeamPCP attack technology clearly demonstrates this dimension rise. In traditional supply chain attacks, attackers often use relatively fixed patterns-such as implanting malicious code in packages, exploiting known configuration vulnerabilities, or obtaining credentials through social workers. These means, while effective, are characteristically fixed and, once analyzed by the community, can be defended by signatures or behavioral baselines. However, TeamPCP the introduction of AI-assisted polymorphic code generation, which allows each attack to use code variants with different characteristics-the same function can be achieved through completely different code structures-traditional signature detection is almost ineffective in the face of this polymorphism.The jump-up dimension of attack capability is reflected in five levels:

- Package manager layer: Attackers are no longer satisfied with simple code implantation, but use AI to generate fake packages that are almost the same as legitimate packages-including package name, README, version number, and some functions. Malicious behavior is only triggered when a specific API is called.
- CI/CD pipeline layer: Attackers use AI to analyze the CI/CD configuration and automatically generate malicious workflow changes that can bypass security detection-these changes are syntactically legal, appear harmless in function, but under certain conditions, leak keys or implant backdoors.
- Cloud environment layer: Attackers use AI-generated IaC (Infrastructure as Code) to code malicious infrastructure, making the creation of malicious resources indistinguishable from normal cloud operations.

- Distribution layer: Attackers use AI to generate a large number of realistic fake Star, Fork and Issue, creating the illusion that the package is widely used and inducing developers to include it in dependencies.

- Trust system layer: The attacker even forges the construction traceability certificate of SLSA Build Level 3, so that the contaminated product passes all security checks in the source verification process.

The causal transmission logic of the attack technology transition is: weakening the effectiveness of detection AI reducing the complexity of the attack threshold → AI the adaptive attack chain → significantly increasing the success rate of the attack → shifting the attack target to a higher value. When the attack technology is strong enough, attackers are no longer satisfied with low-value end targets, but move to higher-value infrastructure credentials and trust anchors.

4.4 Value Layer: The shift in the center of value drives a realignment of core attack objectives.

The value layer consists of elements and variables IV (value center of gravity migration) and is a key turning point in the overall causal architecture. Driven by the first three layers (scenario layer, expansion layer, threat layer), the attacker's target has undergone a high-value migration-from the traditional "terminal data and code assets" to "infrastructure credentials and trust anchors". This shift has not only changed the target of attack, but also more profoundly shuffled the economic model of attack and the mechanism of trust.

Elements and Variables IV: Value Center Migration-Value Transfer from "Terminal Data" to "Infrastructure Voucher. In the traditional network security paradigm, the ultimate goal of the attacker is the end user's sensitive data-credit card number, personal identity information, trade secrets, etc. Therefore, the focus of defense is to protect the end device, the database and the application layer. However, in a supply chain attack, the goal of an attacker has shifted fundamentally: stealing a cloud key may be worth far more than stealing all the user data in a database. Because the cloud key can access all resources in the cloud environment-including other databases, buckets, K8s clusters, and even across cloud accounts. This change in value perception is directly reflected in the priority of stealing IAM role credentials rather than directly stealing business data in TeamPCP attacks.

The logic of this value transfer lies in the leverage effect. The value of terminal data is one-time-it can be sold or extorted after theft, but once it is discovered, the data owner can change the password and revoke the certificate, and the value of the attack disappears. The value of infrastructure credentials is continuous-a stolen GitHub PAT can be used to monitor code changes for a long time, inject malicious code, and even take over the entire code warehouse;

A stolen npm release Token can be used to continuously release malicious package versions; A stolen K8s ServiceAccount Token can be used to reside long-term in the internal network of the enterprise. The voucher is the "master key", and mastering the voucher means mastering the control of the system.

The value center of gravity migration is also reflected in the calculation of attack ROI (return on investment). Attacking an end user needs to customize the attack payload (payload) for each target, which is costly and inefficient. Attacking a widely used open source package (such as axios and React) can affect thousands or even tens of thousands of downstream projects through poisoning at one time. TeamPCP fake (Typosquatting) attacks choose these well-known packages as targets precisely because of their extremely high "impact multiplier"-a successful poisoning can bring exponential benefits. This ROI calculation allows attackers to naturally focus their resources on the "high-value nodes" of the supply chain-packages and platforms with extensive dependencies, high downloads, and high community impact.

The causal transmission of the value layer is bifurcated in both directions: attack target transfer → voucher value exceeds code value, and then it is transmitted in two directions-trust model failure → factor and variable six (trust system collapse), and attack ROI enhancement → factor and variable eight (attack economic expansion). This shows that the value center of gravity migration is not only a problem of target selection, but also a structural change of trust system and economic model.

4.5 Effect Layer: The Collapse of the Trust System Triggers Systemic Cascading Risks

The effect layer consists of factor and variable six (trust system collapse) and factor and variable eight (attack economic expansion), which is the chain reaction of the evolution of the value layer at the macro level. When attackers turn their focus to credentials and trust anchors, the traditional trust framework begins to disintegrate, and the economic drivers of the attack activity form a self-reinforcing closed loop. The effect layer is the most harmful node in the causal system-it indicates that the threat has jumped from a "technical vulnerability" to a "systemic crisis".

Elements and variables six: the collapse of the trust system -- the paradigm shift from "identity cognition" to "behavior perception. The defense line of the software supply chain is based on three trusts: source trust (believing that npm/PyPI packages come from legitimate maintainers), identity trust (believing that GitHub accounts belong to real developers), and environmental trust (believing that CI/CD operates as expected). These three pillars form the cornerstone of supply chain security, but the TeamPCP battle shows that they are being broken down one by one.

Disillusionment of source trust: Attackers use typosquatting, account hijacking, or take over the accounts of legitimate maintainers, making malicious packages look the same as regular packages.

Disintegration of identity trust: Attackers pose as trusted developers through social engineering, credential theft, or AI synthetic false identities.

Breakdown of environmental trust: Attackers contaminate CI/CD configurations or falsify build traceability certificates, making contaminated products easily audited. When these three pillars shake, the entire supply chain will fall into the abyss of "trust desert.

In the face of trust collapse, the logic of defense must be completely reversed. Traditional strategies focus on "identity verification" (checking signatures, credentials, and authors), but when identity information becomes a forgery tool, it must be upgraded to "behavioral insight" (monitoring runtime exceptions, traffic exceptions, and resource abuse). This shift from "identity perception" to "behavior perception" is a fundamental philosophical revolution in supply chain defense.

Elements and variables VIII: Attacking economic inflation-an exponential positive feedback attack spiral. In the traditional model, the cost of attack is linearly proportional to the benefit-the larger the target, the more investment, and the higher the return. But in the AI-driven supply chain ecology, there is an exponential overlap between costs and benefits:

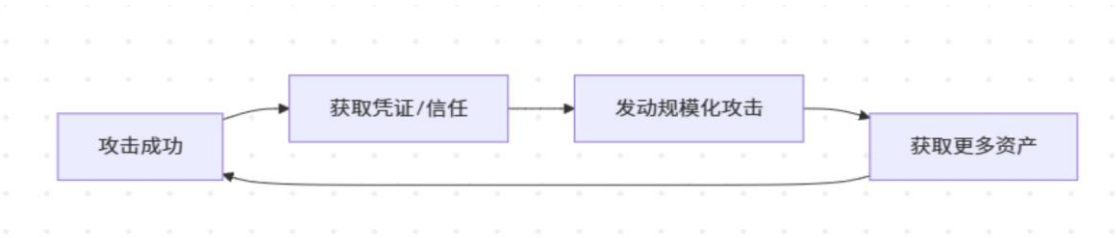


Figure 4-2 Self-accelerating "Attack Flywheel"42

TeamPCP first used social workers to obtain a small number of vouchers and put in the first batch of malicious packages. The chain of infection of these packages, in turn, helps them steal more high-value credentials (such as CI/CD tokens, cloud AccessKey), which in turn hack more repositories. Each successful attack is the "capital" of the next round of expansion, resulting in a geometric increase in attack capability, far exceeding the response speed of the defender. This mechanism gives rise to a "winner-take-all" effect: the group that takes the lead in mastering AI attacks will quickly monopolize high-value assets.

The causal transmission logic of the effect layer is: the failure of the trust model, the collapse of the basic line of defense, the fuzzy boundary of responsibility (pointing to the factor and variable seven), while the ROI is improved, the positive feedback spiral, the asymmetric expansion of the attack input (pointing to the factor and variable nine).

4.6 Defense layer: governance lags in defense give rise to a risk-loop feedback loop

The defense layer consists of elements and variables seven (defense responsibility mismatch) and elements and variables nine (governance and detection lag). Both are the inevitable result of the chain conduction of the first six elements and variables, and are also the core closing link of the closed-loop system of nine elements and variables. As the ecological scale of the software supply chain continues to expand, the complexity of the system continues to increase, the iterative evolution of attack technology and the gradual collapse of the industry trust base, the defense system will continue to fall into a serious responsibility vacuum, governance lag double dilemma. The above two major problems will flow back to the scene layer through the feedback path, resulting in a new forward enhancement loop. Therefore, the defense layer is not the end of the whole system cycle, but the new starting point of a new round of risk evolution and attack and defense cycle.

The core of defense responsibility mismatch is the institutional governance dilemma of "who should be responsible for the security of the software supply chain. The software supply chain link involves multiple responsible subjects, including: open source package maintainers, developers who call open source packages, package management platform service providers, CI/CD tool service platforms, cloud infrastructure vendors, and enterprises that finally complete software deployment and operation and maintenance. All subjects generally have a prevarication mentality of security responsibility and attribute security responsibility to other participants: open source package maintainers believe that they only contribute code free of charge and have no obligation to ensure absolute security of the code; Developers believe that they only use open source packages in compliance with regulations and do not need to check the security of dependent codes line by line. All kinds of service platforms believe that they only provide basic operation and service capabilities and have no obligation to conduct security audits on open source content uploaded by users; landing enterprises believe that commercial safety products have been purchased, and safety protection should be the sole responsibility of safety manufacturers. This lack of collective responsibility for the whole link has created a huge supply chain security vacuum and has become a core breakthrough for attackers to break through defense.

The distribution characteristics of the victims of TeamPCP attacks intuitively confirm the serious consequences caused by the mismatch of responsibilities. The React/Node.js ecological open source package attacked this time is mostly maintained by individual developers or small research and development teams, and lacks the manpower and resources to continuously carry out security iteration and vulnerability maintenance. Enterprise research and development personnel who call such open source packages are highly dependent on the pre-security review of the package management platform, but the platform's audit capability is limited and cannot cover millions of open source packages in the platform. At the same time, the traditional protection products of security vendors mainly deal with application layer attacks, and there are obvious shortcomings in the detection and defense capabilities of new software supply chain attacks. Multi-subject layer of prevarication, passive reliance on others to protect the mentality, and ultimately resulting in the whole link security no one to cover the bottom of the situation.

Elements and variables Seven core feedback mechanisms: fuzzy responsibility boundaries → safety responsibility vacuum → collapse of trust foundation → more fuzzy responsibility boundaries. It's an internal cycle-responsibility mismatch leads to trust collapse, and trust collapse exacerbates responsibility mismatch. When each subject counts on others to take responsibility, attackers exploit this collective absence to "supply chain poisoning". Once the supply chain is compromised (such as malicious package intrusion), the affected enterprises will face a serious data leakage or extortion crisis, which itself will completely destroy the trust foundation of the industry in the open source ecosystem, thus increasing the fragmentation of defense.

Elements and variables nine: governance and detection lag, the core is the system response iteration and attack technology evolution between the "time difference" dilemma. Even assuming that the responsibilities of each subject are clearly divided and all parties actively invest resources to build a defense system, there is still a fundamental lag in the existing supply chain security detection technology and governance mechanism. The iterative speed of detection tools and detection methods on the defensive side lags far behind the evolutionary speed of attack technology. Signature detection cannot identify AI polymorphic code attacks, behavior detection is difficult to identify attacks disguised as legitimate operations, and the vulnerability database matching mechanism is completely ineffective for non-vulnerability attacks. TeamPCP attacks rely on AI-assisted code generation capabilities, which can achieve complete differentiation of the payload (payload) characteristics of each attack, completely circumventing the traditional detection system based on feature matching, resulting in the complete failure of traditional protection methods.

The problem of lagging governance is mainly reflected in the macro-system level. At present, the global mainstream software supply chain security governance frameworks (SLSA, SSDF, NIST supply chain security guidelines, etc.) all take "attacker tampering with software package" as the core presupposition to build the system, while the trust system pollution attacks launched by TeamPCP (such as forging SLSA Build Level 3 compliance certificate) directly subvert the design premise of the existing governance framework. The security standard system itself has become a key breakthrough target for attackers, and the iteration rhythm of existing standards lags far behind the rapid evolution of attack technology. At the same time, supply chain security governance involves multiple complex issues such as cross-border jurisdiction, intellectual property rights, and open source community autonomy, and there is great resistance to the implementation of cross-border collaborative governance. In addition, TeamPCP attacks have the characteristics of global diffusion and global spread, the governance and control measures of a single country can not form effective protection in the whole domain, and the limitations of protection are very prominent.

There is a key feedback path for element and variable nine (dashed line, labeled "attack>> detection" on the right): exponential widening of the gap between attack and defense → ecological lack of security constraints → further ecological expansion (back to element and variable one). The path is a closed loop of nine elements and variables-the defensive lag allows the attacker to act more recklessly, thus exacerbating the expansion of the ecology and the growth of the exposed surface, opening a new round of positive feedback loop.

There is also a direct causal transmission between factor and variable seven and factor and variable nine: no overall defense (output of factor and variable seven) → exponential widening of the gap between attack and defense (input of factor and variable nine). The mismatch of responsibilities leads to the fragmentation of defense, the fragmentation of the point defense can not build the whole domain linkage of the overall defense line, and the absence of the overall defense line, and ultimately lead to the gap between attack and defense technology continues to widen, the imbalance between attack and defense situation intensified.

5 Scenario-based defense checklist

This chapter takes the results of the previous asset combing and attackable surface risk analysis, and refines five sets of standardized landing checklists around the high-frequency landing scenario of the open source software supply chain. According to the new project launch, environmental governance, container deployment, open source access, emergency disposal scenarios, each content clearly check the action, acceptance criteria, the list can be directly

implemented, to help enterprises to turn the security strategy into a landing normal self-examination action. Based on the previous threat characteristics and the conclusion of the full link attack surface, the abstract security requirements are implemented as five business scenario checks, each with a matching verification method and eligibility criteria. The security team can be selected according to its own R & D stage for on-line acceptance, daily inspection, incident handling, etc.

5.1 Scenario 1: Checklist for the initial startup of cloud-native applications

For the scenarios of new project initiation, warehouse initialization, and new environment deployment, the following table lists the key check items from private source configuration, dependency locking to container signature, monitoring access, etc. to ensure that the application meets the security baseline from "birth.

Table 5-1 Cloud Native Application New Startup Security Checklist51

Number	Check item	Method of operation	Passing standard
1	Private source configuration control	Configure. npmrc and pip.conf to point to internal private sources	Development environment cannot directly connect to public package source
2	Dependent Version Locking	npm lock, Poetry Lock Generate Lock File	Lock file into code control
3	CI Access Vulnerability Scan	Snyk, Dependabot Integration Pipeline	Code entry automatically triggers dependency risk detection.
4	Submission Signature Control	Configuring GPG Force Commit Signatures	All code submissions carry valid signatures
5	Warehouse branch protection	Configure Branch Protection Policy	Code changes must be subject to PR manual review
6	Key Centralized Escrow	Unified storage of credentials into key management platform	Code no plaintext hard-coded key
7	Workflow change control	workflows Directory Incorporates Branch Protection	Pipeline configuration modification requires approval
8	Project SBOM Generation	Syft Generate Bill of Materials in SPDX format	SBOM unified archive management with code
9	Container image signature	Cosign Embedded Image Build Process	Image Push Auto-Complete Signature Verification
10	K8s baseline configuration	Enable PSS, Network Security Policy	New Pod Adhere to Cluster Security Baseline

11	AI coding specification landing	Develop a system for the use of internal AI tools	Full implementation of sensitive code prohibition requirements
12	Container operation monitoring	Cluster deployment Falco	Real-time alarm for abnormal container behavior
13	Cloud Voucher Lightweight	Replacing static keys with OIDC and IAM roles	Long-term AK/SK does not exist in the environment
14	Operation Log Collection	Unified Log Collection Solution	Full link operation traceable audit

5.2 Scenario 2: Unified security governance checklist for multi-cloud environments

For the overall management requirements of multi-account and cross-cloud architecture, the following table provides verification points such as root account reinforcement, metadata protection, and storage permission control to support normalized security governance in multi-cloud environments.

Table 5-2 Unified Security Governance Checklist for Cloudy Environment2

Number	Check item	Key points of verification	Passing standard
1	Root account reinforcement	Open MFA for all Root accounts and prohibit daily business login	Multi-factor authentication has been enabled for all cloud root accounts, which are only used for emergency operations and have no daily business login records.
2	Metadata protection	Enable full IMDSv2 protection for cloud services	Disable IMDSv1 for all cloud servers and container instances and retain only the IMDSv2 access mechanism to prevent metadata theft.
3	Storage permission control	The object bucket is private by default, and the full-network public permission is disabled.	All object buckets have no public read or write permissions, and only the intranet and specified business accounts are authorized to access them.
4	security group convergence	Follow the minimum open port configuration principle	The security group can only open the port and IP whitelist required for business, and does not have the configuration of opening all ports and all network segments.
5	Enable audit log	Normalization of the whole platform operation audit log is opened	The audit log of the multi-cloud platform is continuously opened, and the log is completely retained and cannot be deleted or tampered with at will.
6	Configuration Inspection Landing	CSPM covers all cloud resource accounts	CSPM tools cover all cloud accounts and resources, and regularly carry out configuration risk inspection and closed-

			loop rectification.
7	Cross-account permission control	Delineate cross-cloud access trust boundaries and strictly control ultra vires channels.	Cross-account and cross-cloud access permissions are clear, without anonymous access or over-authorization channels.
8	Unified IAM Templates	Standardize permission templates to avoid permission flooding configuration	All employees and all business accounts uniformly reuse standardized IAM permission templates, without custom super-large permission configuration

5.3 Scenario 3: Security checklist for containerized deployment

Before the image is launched and the K8s service is released, the following table provides specific verification methods and eligibility criteria for non-root operation, permission clipping, image signature, network policy, and access control to prevent unsafe and insecure containers from entering the production environment.

Table 5-3 Security Compliance Checklist for Containerized Deployment5-3

Number	Check item	Verification method	Passing standard
1	Container Run Identity	Verify Dockerfile and container security context	The container is not allowed to run with root
2	System privilege clipping	Combing container Linux kernel Capability permissions	Retain only business-necessary permissions
3	Foundation Mirror Selection	Source Trace Mirror Source	Optimization of Lightweight Security Foundation Image
4	Mirror Vulnerability Scan	View CI Scan Report	High-risk vulnerability image interception release
5	Image signature verification	Configure Pull Validation Rules	Forced verification of image signatures before going online
6	Cluster Key Encryption	Verify the etcd encryption configuration	K8s Secret encrypted storage at rest
7	Cluster Network Policy	Namespace Configuration NetworkPolicy	By default, cross-pod unauthorized access is denied
8	Access control landing	Deployment Gatekeeper/Kyverno	Violation Pod cannot complete creation
9	Helm package verification	Chart Publishing Verification Source and Signature	Deploy Trusted Source Template Packages Only
10	resource quota	Configure the upper and lower	All Pods Configuration Resource Limit

	constraint	limits of CPU and memory	

5.4 Scenario 4: security assessment checklist for open-source component admission

When introducing new open source packages or third-party tools, the following table provides evaluation criteria and rejection red lines in terms of project activity, maintainer background, historical vulnerabilities, and installation script audit to implement security access control for open source components.

Table 5-4 Open Source Component Access Security Assessment Checklist4

Number	Check item	Verification content	Reject Red Line
1	Project Maintenance Activity	View code updates and problem responses	Over 1 year without any version iteration directly rejected
2	Maintenance Party Background Verification	Check developer history project information	New Empty Account No History Items Not Accused
3	Project use scale	Statistics weekly downloads and associated items	Very low usage cautiously introduced
4	Historical Vulnerability Troubleshooting	Retrieving Historical CVE Based on Vulnerability Library	High-risk vulnerabilities that have not been fixed are not allowed to enter.
5	Installation Script Audit	Key check preinstall hook code	Existence of outreach, theft class logic disabled
6	Dependency Link Control	Combing the number of indirect dependencies	Reliance on hierarchy over-expansion Prudent access
7	Open Source License Validation	Compare License Compatibility Rules	License conflicts prohibit the introduction
8	security response mechanism	Confirm SECURITY.md and vulnerability escalation channels	No safe disposal mechanism to exclude access
9	Open Source Security Score	OpenSSF Scorecard Score	If the score is too low, no purchase will be made.
10	Troubleshooting Naming Conflicts	Compare internal private package name	There is a risk of confusion over the same name. Reject the introduction.

5.5 Scenario 5: Emergency response checklist for supply chain security incidents

In the event of malicious poisoning, vulnerability outbreak, or suspected intrusion, the following table is used to implement the disposal matters in stages, such as warning judgment, risk containment, hidden danger eradication, business recovery, and resumption optimization.

Table 5-5 Supply Chain Security Incident Emergency Response Checklist5

Phase	Number	Disposal matters
Alarm judgment	1	Identify the authenticity of alarms and eliminate false alarms
	2	Delimitation of affected projects, cluster scope
	3	Initiate internal incident notification process
Risk containment	4	Suspend contaminated package build and release
	5	Bulk logout of leaked credentials, tokens
	6	Pull black malicious domain name and C2 address
Eradication of hidden dangers	7	Replace malicious dependencies and clean up backdoor code.
	8	Fix configuration defects and attack portals
	9	Full rotation of risk account keys
Business Recovery	10	Multiple rounds of security scanning to verify the repair effect
	11	Grayscale batch recovery business online
	12	Short-term improvement of full-link security monitoring efforts
Review and optimization	13	The whole team resumed the root cause and summarized the attack path.
	14	Iterative refinement of existing security inspection specifications

This chapter relies on the previous threat study and attack surface combing conclusions, combined with the recent supply chain bulk poisoning of the offensive and defensive characteristics, around the development of five core scenarios to develop a standardized checklist. The supply chain security protection requirements are landed as directly executable verification steps, covering the whole process of new project construction, container deployment, open source access, environmental governance and emergency disposal, realizing the transformation of security strategy from theoretical analysis to practical implementation, and forming a complete protection closed loop of pre-fortification, in-process control and post-disposal.

6 Summary

After the TeamPCP sandstorm supply chain poisoning, what is exposed is the huge but not solid foundation of the global software tool chain system. The world, from software ecology to artificial intelligence research and

development, is a forest of sand dunes that continue to pile up on this foundation. As the "sand worms" continue to cross, they are facing the risk of collapse. The security risk style in the era of artificial intelligence is predicted, conjectured and feared by many people. However, when the first attack wave came, it was not the escape of "I Robot", but the awakening of Skynet like Terminator. At the entrance of each attack, it seems to be no different from a "traditional" cyber attack. exposure discovery, vulnerability exploitation, worm propagation, Trojan implantation, credential theft, but these malicious codes are written with AI assistance. Although the digital world of mankind is constantly evolving, it is a world full of "established facts. Both the consumer and the attacker must complete the link in the way they run the coupling. However, if you take a global view of this sandstorm poisoning, you will find that it is by no means a simple large-scale worm scanning and hit delivery. It is a AI-driven composite killing network that can target every discoverable exposure entrance in the supply chain system. It shows that the attacker not only keeps peeping into the value system of artificial intelligence, but also "embraces" the full empowerment of artificial intelligence far deeper and more firmly than the defender. It is therefore possible to carry out complex strikes on complex targets.

The unusually large and complex development tool supply chain ecosystem is the loot in the eyes of attackers, full of "opportunities and entrances" due to complexity, and is a long position that defenders need to defend. Therefore, we must try to draw a panoramic map of the developer tool chain and sort out the exposed surfaces in detail. We must have a deep understanding of this complex defensive position in order to better assist users and ecological partners in providing defense. In order to better enhance the ability of security engines to serve ecological partners, relying on smart bodies and platforms to empower customers.

Where the threat extends, Antiy's detection, analysis and defense capabilities extend. This is what we have been insisting on. We have moved the anti-virus capability from the host side to the backbone side (2001), we have extended threat analysis to the industrial system (2008), we have basically covered the anti-virus engine to the domestic mobile phone scene, and today we will continue to complete our mission in the era of artificial intelligence.

[Note]: This report contains some artificial intelligence generation content. Analysts rely on Antiy AVL Code AI agent and access Landi VILLM to assist in generation. All model-assisted output content has been manually reviewed, reviewed and proofread.

This report contains some content generated by AI. The analysts relied on the Antiy AVL Code AI agent and accessed the LanDi VILLM to assist in generating the content. All the content produced by the model assistance has undergone manual review, verification and proofreading.

7 Appendix I: Security Checklist, Glossary of Terms, List of Security Frameworks, etc.

A- Supply Chain Security Checklist

This appendix summarizes the full-text research results and compiles them to form a security check list of open source software supply chain tool components, which can be downloaded and used by R & D, security operation and maintenance personnel. The list is divided into worksheets according to modules such as tool component inventory, exposed surface investigation, scenario defense, emergency disposal, governance expansion, etc., covering the whole life cycle safety verification matters, and modules can be selected as required to carry out self-examination. The list supports electronic editing or paper check for landing verification. High-risk items have been marked with stars. It is suggested to retest and rectify periodically on a monthly basis to support the landing of normal supply chain risk control of enterprises.

Download address: http://www.antiy.cn/download/opensource_tool_list.xlsx

开源软件供应链工具组件安全核查清单

使用说明
1. 本工作簿整合了报告中全部检查清单，供安全运维团队日常使用
2. 每个Sheet对应一个检查领域，可根据实际工作需求选择使用
3. 检查项中的 □ 列供勾选使用（打印后手工勾选或电子编辑）
4. 建议每月定期复查，发现风险及时整改
5. 带 ★ 标记的为高优先级检查项，建议优先完成

工作表索引

工作表名称	检查领域	检查项数量	适用场景
工具链组件检查	13大类68+组件类型	22项	全面梳理环境组件资产
攻击暴露面	供应链攻击面	80项+	逐项排查潜在风险入口
场景化防御	五大典型场景	60项+	针对具体场景的落地检查
应急响应	事件响应与排查	40项+	遭遇攻击后的快速响应
扩展清单	DevSecOps/演练/灾备	50项+	安全治理成熟度提升

版本: v1.0 | 发布: 2026年6月 | 维护: 安天CERT

B- Quick Reference Guide to Key Terms

Terminology	Definition
SBOM	Software Bill of Materials, Software Bill of Material
SLSA	Supply-chain Levels for Software Artifacts
SCAaaS	Supply Chain Attack as a Service, Supply Chain Attack as a Service
OIDC	OpenID Connect, open authentication protocol
Cosign	Sigstore tools for container image signing and verification
Typosquatting	Register a name similar to a popular package for attack
IMDS	Instance Metadata Service, cloud instance metadata service
CSPM	Cloud Security Posture Management, cloud security situation management
Key Management Service	Key Management Service, Key Management Service
RBAC	Role-Based Access Control

C- List of Security Frameworks, Regulations, and Intelligence Reference Sources

Resource Type	Name	Link/Source
Laws and regulations	the People's Republic of China Cybersecurity Law, the People's Republic of China Data Security Law, the People's Republic of China Personal Information Protection Law, the People's Republic of China Cryptography Law	chinese legal system
National standard	Equal protection 2.0 (GB/T 22239-2019)	network security level protection
National standard	GB/T 39276-2020	General requirements for security of network products and services
National standard	GB/T 43698-2024	Software Supply Chain Security Requirements
Security Framework	SLSA (slsa.dev)	Software Product Supply Chain Hierarchy
Security Framework	SSDF (NIST SP 800-218)	Secure Software Development Framework
Security Framework	OpenSSF Best Practices	bestpractices.openssf.org
Standard	CIS Benchmarks	cisecurity.org

Framework		
Threat intelligence	CVERC	National Computer Virus Emergency Response Center
Threat intelligence	OpenSSF CVE Feed	Open Source Security
Community	OpenSSF	Open Source Security Foundation www.openssf.org
Security Announcement	GitHub Security Advisories / npm Security Advisories	Platform Security Bulletin
Research Report	Cloud Security Alliance	Cloud Security Alliance

8 Appendix II: References

[1] Antiy CERT-Sandstorm-style Poisoning Targeting Developer Tool Supply Chains - Sample, Technical, and Tactical Analysis of TeamPCP Organization (Part I) (2026-05-27).
https://www.antiy.cn/research/notice&report/research_report/TeamPCP_Shai_Hulud_202605.html
[2] Antiy CERT-Why the TeamPCP "Russian Roulette" module is a false flag (2026-05-27).
https://www.antiy.cn/research/notice&report/research_report/TeamPCP.html
[3] Cloud Security Alliance - Shai-Hulud/Megalodon: A Two-Wave AI Developer Supply Chain (2026-05-23).
https://labs.cloudsecurityalliance.org/research/csa-research-note-shai-hulud-megalodon-supply-chain-cascade/
[4] SafeDep - Megalodon: Mass GitHub Repo Backdooring via CI Workflows (2026-05-18).
https://safedep.io/megalodon-mass-github-repo-backdooring-ci-workflows
[5] Phylum - Shai-Hulud: A Self-Replicating AI Worm in npm (2026-05-20).
https://blog.phylum.io/shai-hulud-a-self-replicating-ai-worm-in-npm/
[6] Socket - React2Shell: AI-Generated React Component with Hidden Backdoor (2026-05-25).
https://socket.dev/blog/react2shell-ai-generated-react-component-hidden-backdoor
[7] Phylum - LiteLLM Cross-Ecosystem Supply Chain Attack (2026-05-22).
https://blog.phylum.io/litellm-cross-ecosystem-supply-chain-attack/
[8] ReversingLabs - Miasma: AI-Powered Python Package with Polymorphic Payload (2026-05-27).
https://www.reversinglabs.com/blog/miasma-ai-powered-python-package-polymorphic-payload
[9] OpenSSF - S2C2F: Open Source Consumption Framework (2026-05-15).
https://openssf.org/projects/s2c2f/
[10] Sonatype - State of the Software Supply Chain 2024.
https://www.sonatype.com/state-of-the-software-supply-chain
[11] SLSA - Supply-chain Levels for Software Artifacts (v1.1).
https://slsa.dev/
[12] NIST - Cybersecurity Supply Chain Risk Management (SP 800-161r1).
https://csrc.nist.gov/publications/detail/sp/800-161/rev-1/final
[13] NIST - Secure Software Development Framework (SSDF) SP 800-218.

https://csrc.nist.gov/publications/detail/white-paper/2023/04/11/secure-software-development-framework/final
[14] CISA - Software Supply Chain Security Guidance.
https://www.cisa.gov/software-supply-chain-security
[15] OpenSSF - Scorecard: Security Health Metrics for Open Source Projects.
https://securityscorecards.dev/
[16] SPDX - Software Package Data Exchange Specification.
https://spdx.dev/
[17] CycloneDX - Bill of Materials Standard.
https://cyclonedx.org/
[18] OWASP - Dependency-Check: Software Composition Analysis.
https://owasp.org/www-project-dependency-check/
[19] GitHub Advanced Security - Secret Scanning and CodeQL.
https://github.com/features/security
[20] BSIMM - Building Security In Maturity Model (Software Security Framework).
https://www.bsimm.com/
[21] OWASP SAMM - Software Assurance Maturity Model.
https://owaspsamm.org/
[22] Microsoft - S2C2F: Secure Supply Chain Consumption Framework.
https://github.com/microsoft/S2C2F
[23] Endor Labs - Dependency Management Best Practices.
https://www.endorlabs.com/learn/dependency-management-best-practices