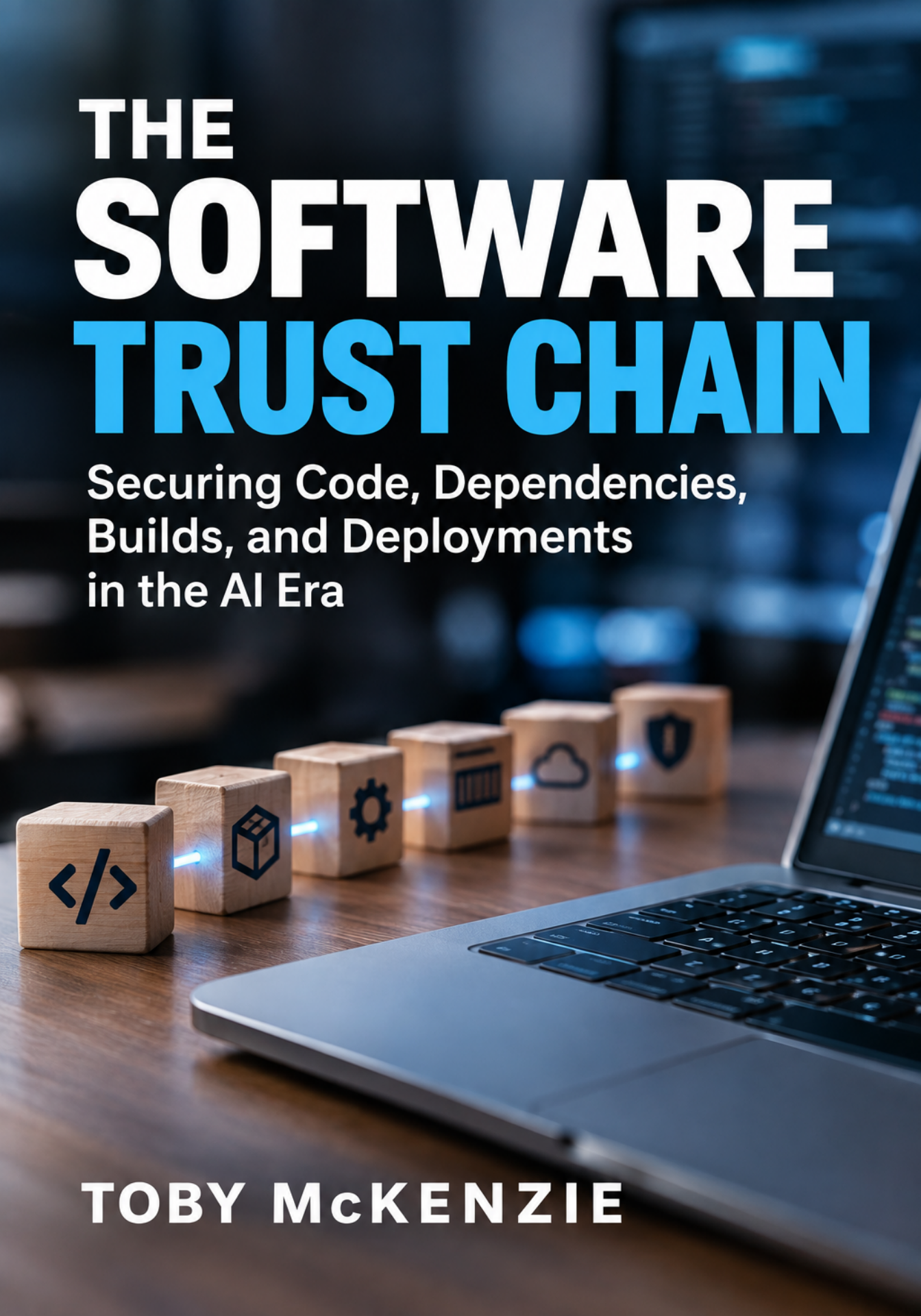
THE SOFTWARE TRUST CHAIN

Securing Code, Dependencies, Builds, and Deployments in the AI Era



TOBY McKENZIE

The Software Trust Chain

Securing Code, Dependencies, Builds, and Deployments in the AI Era

Steve Publications

This book is available at https://leanpub.com/thesoftwaretrustchain

This version was published on 2026-09-20

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Securing Code, Dependencies, Builds, and Deployments in the AI Era

This book provides a comprehensive technical reference on securing the modern software supply chain, from source code through dependencies, build systems, CI/CD pipelines, containers, infrastructure as code, and production deployment, with particular attention to how generative AI and autonomous coding agents transform the threat landscape. It is written for experienced software engineers, DevOps and platform engineers, security engineers, architects, and engineering managers who need deep understanding of mechanisms, trade-offs, and implementation strategies rather than high-level summaries or tool surveys. The goal is to make the material durable: by focusing on principles, architectures, and engineering practices, this book will remain useful even as individual tools and platforms evolve.

Introduction

Every modern software release is an act of faith. When an engineer pushes code and a build pipeline runs, dozens of invisible systems participate: version control servers authenticate the commit, package managers resolve hundreds of dependencies, build tools fetch their own dependencies, CI runners download container images, artifact registries store the output, deployment tools verify credentials, and orchestration platforms pull images from the network. None of this is visible to the developer pressing the merge button. The code that ships is the product of a vast, distributed, largely unexamined process.

That process is the software supply chain, and it has become the single largest attack surface in modern software engineering. Attacks do not need to break into your network or guess your passwords. They can compromise a package you depend on, tamper with a build tool, modify a CI/CD configuration, poison a container image, or manipulate an AI coding assistant into suggesting a malicious dependency. The attack happens before your software even exists.

The problem is not new, but its scale is. A decade ago, most applications were built primarily from code written in-house, with a handful of external libraries. Today, open source constitutes an estimated seventy to ninety percent of the code in an average software project [1]. The average application depends on roughly one hundred fifty to one hundred eighty open source components, and ninety percent of those are indirect dependencies that no developer in the organization has ever reviewed [2]. Meanwhile, malicious packages across all ecosystems now number in the hundreds of thousands and are growing at rates exceeding one hundred fifty percent year over year [3].

Generative AI adds a new dimension to an already complex problem. Coding assistants routinely suggest dependencies that do not exist, invent plausible-sounding package names that attackers can register with malicious payloads, and generate code containing security weaknesses at measurable rates [4, 5]. Autonomous agents that can edit files, run commands, and interact with repositories and CI/CD systems introduce execution-layer risks that did not exist when AI merely suggested code for humans to review.

Traditional application security was designed for a simpler world. Static analysis scans code for known vulnerability patterns. Dynamic analysis probes running applications for exploitable flaws. Penetration testing simulates attacks against deployed systems. None of these approaches can protect you when the attacker never touches your code, your network, or your deployed software. The compromise happens upstream, in the infrastructure and dependencies you rely on to build and deliver that software.

This book treats software supply-chain security as an engineering discipline, not a policy exercise. It assumes that readers understand software development and infrastructure and want to know how the supply chain works, where it fails, what can be done about it, and what the trade-offs are. The material progresses from foundations through threats, defensive architectures, implementation strategies, organizational practices, real-world incidents, and finally to the challenges posed by AI. The goal is not to produce a checklist but to build understanding deep enough to make sound decisions when there is no perfect answer.

The central argument is straightforward: modern software supply chains are too complex and too distributed for perimeter-based or point-in-time security controls. Protecting them requires a fundamentally different model: verifiable provenance, strict identity boundaries, policy enforcement at every transformation step, continuous validation rather than one-time checks, and acceptance that some risks must be managed rather than eliminated. AI accelerates both the scale of the problem and the potential for automated defense, but careless adoption of AI in development workflows risks inheriting and amplifying all existing supply-chain weaknesses at machine speed.

This book does not attempt to survey every tool or vendor. The supply-chain security ecosystem moves too quickly for that approach to remain useful beyond a few months. Instead, it focuses on the principles, mechanisms, and architectures that endure, using current technologies as examples rather than prescriptions.

The reader who completes this book will understand the end-to-end journey of software from source to runtime, the attacks that target each stage, the defensive controls that address them, the standards and frameworks that provide structure, the organizational practices that make implementation sustainable, the lessons from real-world breaches, and the new challenges introduced by AI-assisted and AI-autonomous development. Most importantly, they will be able to evaluate their own supply chain, identify the most conse-

quential risks, and make informed trade-offs between security, velocity, and operational complexity.

Chapter 1: What Is a Software Supply Chain

The first step in securing anything is defining what "it" is. The phrase software supply chain is used broadly and often loosely, which creates confusion about scope and responsibility. This chapter establishes a precise definition, traces the journey of software through the supply chain, explains why traditional application security falls short, and sets up the mental model that the rest of the book depends on.

The Invisible Infrastructure Behind Every Release

When a developer writes code and merges it into a repository, they are initiating a process that extends far beyond their local machine. That code will be fetched, combined with other code, built, tested, signed, stored, deployed, and eventually executed, often on systems owned by third parties. Each step relies on infrastructure and services that are themselves software, with their own dependencies, their own vulnerabilities, and their own supply chains.

Consider a simple example. A Node.js developer creates a feature branch, writes code that uses express and a few other libraries, and opens a pull request. The repository lives on GitHub. The pull request triggers a GitHub Actions workflow that runs tests in an Ubuntu container, installs dependencies from npm, builds a Docker image, pushes it to GitHub Container Registry, and if all checks pass and a maintainer approves, merges into the main branch. Another workflow deploys the image to a Kubernetes cluster.

How many trust relationships exist in that process? The developer trusts GitHub to store the code and run the workflow. The workflow trusts the Ubuntu container image to be unmodified. It trusts npm to deliver the correct versions of express and its transitive dependencies. It trusts Docker to build the image correctly. It trusts the container registry to store and serve the image without tampering. The Kubernetes cluster trusts the registry to deliver the image at deployment time and trusts the container runtime to execute it faithfully.

Each link in this chain represents a potential attack surface. A compromised Ubuntu base image delivers a backdoor to every build that uses it. A malicious npm package exfiltrates secrets from the CI environment. A tampered container image runs arbitrary code in the cluster. The developer wrote no vulnerable code, reviewed no suspicious dependency, and clicked no phishing link. The attack succeeded because the supply chain was compromised at a point where no one was looking.

This is not hypothetical. The incidents documented throughout this book demonstrate that attackers have targeted and successfully exploited every stage of the supply chain described above. The question is not whether supply-chain attacks exist but whether an organization is willing to accept the risk that its software depends on infrastructure and dependencies it cannot fully control.

From Source to Runtime: A Complete Journey

To understand the software supply chain, we need to map the complete journey of software from source to runtime. The following stages represent the canonical path, though real-world implementations may add, remove, or reorder steps. Each stage will be examined in detail in subsequent chapters.

The journey begins with source code and version control. Modern development relies on distributed version control systems, primarily Git, hosted on platforms such as GitHub, GitLab, or Bitbucket. These systems store the authoritative copy of code, manage access control, enable collaboration through branches and pull requests, and provide the foundation for automation. Source control is both the starting point of the supply chain and a high-value target for attackers.

Next comes dependencies. Almost no modern application is built solely from source code in its own repository. Developers depend on libraries, frameworks, build tools, and testing frameworks, fetched from package managers and registries like npm, PyPI, Maven Central, RubyGems, NuGet, and Cargo. These dependencies are themselves composed of further dependencies, creating deep transitive dependency graphs. When a developer specifies a dependency, they are implicitly trusting the publisher, the package, the build environment that produced it, and the registry that serves it.

The build phase transforms source code and dependencies into deployable artifacts. Build tools like make, gradle, maven, webpack, or cargo resolve

dependencies, compile code, bundle assets, and produce binaries, containers, or other deliverables. Build environments introduce their own trust requirements: the operating system, the toolchain, the plugins and extensions, the cache directories. A compromised build tool can inject malicious code into every artifact it produces. A poisoned build environment can tamper with artifacts without modifying source code.

CI/CD pipelines automate the build and delivery process. Tools like GitHub Actions, GitLab CI, Jenkins, CircleCI, and others execute workflows triggered by code changes, running tests, performing security scans, building artifacts, and deploying to environments. Pipelines orchestrate the supply chain, connecting source control, build systems, artifact registries, and deployment targets. They are also a concentration of risk: they often run with elevated permissions, they have access to secrets, and they execute code from external sources, including third-party actions and plugins.

Containers have become the dominant packaging format for cloud-native applications. Tools like Docker build images by layering filesystem changes on top of base images, producing portable, immutable artifacts that can be deployed consistently across environments. Container registries like Docker Hub, GitHub Container Registry, and private registries store and distribute images. Base images are themselves a supply-chain component, often pulling in dozens or hundreds of OS packages. A compromised base image or a tampered image in a registry affects every application that uses it.

Infrastructure as code (IaC) defines the environments where software runs. Tools like Terraform, Pulumi, CloudFormation, and Kubernetes manifests specify networking, compute, storage, databases, and access controls. IaC introduces its own supply chain: modules, providers, plugins, and external data sources. A malicious Terraform module can provision misconfigured infrastructure or exfiltrate credentials. A compromised provider can modify the behavior of infrastructure deployments.

Deployment brings software and infrastructure together. Deployment tools push images and configurations to production environments, where orchestration systems like Kubernetes schedule workloads, service meshes manage traffic, and application load balancers expose services. The trust chain must extend into production: deployed artifacts must be what was built and tested, and runtime environments must enforce security policies.

Finally, software runs in production, where it interacts with users, data, and

other services. Runtime security protects against exploitation of vulnerabilities that survived all upstream controls. Monitoring, logging, and observability provide visibility into whether software behaves as expected. Runtime is not the end of the supply chain: it is the final stage where supply-chain compromises manifest as observable events.

Understanding this journey is essential because every stage is a potential attack surface and every stage is a potential control point. Supply-chain security is not a single control applied at one stage but a coordinated set of controls applied across the entire journey.

Why Application Security Is Not Enough

Application security has been a recognized discipline for decades. SAST tools scan source code for known vulnerability patterns like SQL injection, cross-site scripting, and buffer overflows. DAST tools probe running applications for exploitable flaws. SCA tools scan dependencies for known vulnerabilities. Penetration testing simulates attacks to identify weaknesses. These techniques remain valuable, but they are insufficient for supply-chain security for several reasons.

First, application security focuses on code, not process. SAST examines source code for patterns that indicate vulnerabilities. It cannot tell you whether the compiler that produced the binary was modified to inject a backdoor. It cannot tell you whether the CI/CD pipeline was configured to upload artifacts to an attacker-controlled server. It cannot tell you whether the deployment tool was compromised to replace a legitimate container image with a malicious one. The vulnerability is not in the code but in the process that transforms code into deployed software.

Second, application security is inherently reactive. SAST tools look for patterns associated with known vulnerability classes. They cannot detect a novel supply-chain attack that does not match an existing pattern. DAST tools test for exploitable flaws in a running application, but they cannot detect that the application was built from a compromised dependency or deployed from a tampered image. Penetration testing occurs at a point in time and may miss vulnerabilities that are introduced after the test.

Third, application security assumes that the code under test is the code that will run. In a secure supply chain, this assumption holds: the source code

is built in a trusted environment, the resulting artifacts are integrity-protected, and the deployed software is exactly what was built and tested. In an insecure supply chain, the code in the repository may differ from the code that is built, the build may produce different artifacts than expected, or the deployment may substitute a different artifact than was tested. Application security tools operating on the source code cannot detect these discrepancies.

Fourth, application security treats dependencies as inputs, not as processes. SCA tools compare dependency versions against vulnerability databases, but they do not verify how those dependencies were built, whether the registry was compromised, or whether the package manager resolved to the intended package. A dependency may have no known vulnerabilities in its code but still be malicious because it was tampered with after publication or because an attacker published a lookalike package with a similar name.

None of this means that application security is obsolete. SAST, DAST, SCA, and penetration testing remain essential controls that should be part of any comprehensive security strategy. But they are necessary, not sufficient. Supply-chain security requires additional controls that protect the process of building, testing, and deploying software, not just the code itself.

Trust in a Distributed World

The software supply chain is built on trust. Developers trust that the code in a dependency package is what the publisher claims it is. Teams trust that the CI/CD pipeline executes their workflows faithfully. Organizations trust that the container images in a registry are genuine. None of this trust is cryptographic by default. It is based on assumptions about platform integrity, access controls, and operator honesty.

Modern software development has distributed trust across a vast ecosystem. A single application may depend on code maintained by hundreds of individuals across dozens of organizations, built with tools produced by yet more organizations, running on infrastructure owned by cloud providers, and deployed to environments shared with other tenants. Each link in this chain introduces a trust assumption. If any assumption fails, the software is compromised.

The challenge is not that trust is misplaced. The challenge is that the scale and opacity of modern supply chains make it impossible to validate every

trust assumption manually. An organization cannot audit every dependency maintainer, inspect every build environment, or monitor every registry. The only practical approach is to reduce the number of trust assumptions through cryptographic verification, automate the validation of the remaining assumptions through tools and processes, and design systems so that a single point of trust failure does not compromise the entire supply chain.

This is the foundation of supply-chain security: treating trust as a first-class engineering concern, making it explicit rather than implicit, verifying it cryptographically wherever possible, and designing architectures that survive individual trust failures.

The Cost of Getting It Wrong

Supply-chain attacks are attractive to attackers because they offer outsized returns on investment. Compromising a single dependency, build system, or deployment tool can affect thousands or millions of downstream targets simultaneously. The SolarWinds attack affected approximately eighteen thousand customers through a single compromised software update [6]. The XZ Utils backdoor, discovered in March 2024, could have provided remote code execution on hundreds of millions of Linux servers worldwide [7]. These are not theoretical impacts: they represent real attacks that succeeded by exploiting supply-chain weaknesses.

The cost of supply-chain breaches extends beyond immediate technical damage. Remediation requires identifying all affected systems, revoking compromised credentials, rebuilding software from verified sources, and potentially rerolling deployments. Downtime during remediation can disrupt business operations. Legal and regulatory exposure increases when sensitive data is compromised through a third-party vendor. Trust with customers and partners erodes when an organization cannot demonstrate that its software is secure.

For open-source projects, the costs are even more asymmetric. Most maintainers contribute in their spare time, with no organizational backing, no dedicated security resources, and no incident response capability. Yet their software is used in mission-critical systems by organizations that have no direct relationship with the maintainer. When a maintainer is targeted, the impact propagates through the entire ecosystem, while the maintainer

bears the reputational and operational cost. This asymmetry is one of the fundamental tensions in modern software supply-chain security.

How This Book Is Organized

This book is structured to build understanding progressively, starting with foundations and moving through threats, defenses, implementation, operations, and future developments.

Chapter 2 examines how we arrived at the current state: the historical trajectory from monolithic applications to dependency-heavy, cloud-native, AI-assisted development. Understanding the evolution of the software supply chain clarifies why it is so complex and why traditional security approaches are insufficient.

Chapter 3 establishes the foundational security principles that recur throughout the book: zero trust, least privilege, defense in depth, assume breach, separation of duties, and treating security as an engineering discipline rather than a policy exercise. These principles provide the vocabulary and conceptual framework used in subsequent chapters.

Chapters 4 through 10 map the software supply chain end-to-end, examining each stage from a security perspective. Chapter 4 covers source control and code ownership. Chapter 5 covers dependency ecosystems. Chapter 6 covers build systems and artifacts. Chapter 7 covers CI/CD pipelines. Chapter 8 covers containers and image supply chains. Chapter 9 covers infrastructure as code. Chapter 10 covers deployment and runtime environments. Each chapter explains what can go wrong, how attackers exploit the weaknesses, and what controls exist.

Chapters 11 through 15 examine major attack classes in depth, organized by the stage they target. Chapter 11 covers package attacks: typosquatting, dependency confusion, malicious packages, compromised maintainers, and dependency substitution. Chapter 12 covers repository and code attacks: poisoned commits, malicious pull requests, and repository compromise. Chapter 13 covers build and pipeline attacks. Chapter 14 covers container and infrastructure attacks. Chapter 15 covers secrets, keys, and identity. Each chapter includes detailed technical analysis of real-world incidents.

Chapters 16 through 19 examine the defensive technologies and frameworks that address supply-chain risks. Chapter 16 covers provenance and attestation.

Chapter 17 covers Software Bills of Materials. Chapter 18 covers digital signatures and trust infrastructures like Sigstore. Chapter 19 critically examines SLSA and other security frameworks, explaining what they do, what they do not do, and how to use them pragmatically.

Chapters 20 through 22 address the transformation caused by AI. Chapter 20 covers AI-assisted development and the risks introduced by coding assistants. Chapter 21 examines autonomous coding agents and the new security questions they introduce. Chapter 22 covers AI models, plugins, and datasets as supply-chain components.

Chapter 23 provides reference architectures for different types of organizations, showing how security controls can mature incrementally. Chapter 24 offers practical guidance for assessing and hardening an existing supply chain. Chapter 25 covers operations, monitoring, and incident response. Chapter 26 addresses governance and organizational structure.

Chapter 27 confronts the trade-offs between security, velocity, usability, and cost, providing frameworks for making informed decisions. Chapter 28 looks forward at how supply-chain security must evolve as AI agents become more autonomous and development workflows change.

The book assumes technical fluency: readers should understand software development, version control, build systems, cloud infrastructure, and networking. Specialized security concepts are explained when first introduced, but the material does not begin at a beginner level. The goal is a comprehensive technical reference that engineers can use to design, implement, and operate secure software supply chains.

Chapter 2: How We Got Here

To understand why the software supply chain is so complex and so vulnerable, we need to trace how it evolved. Modern supply-chain architecture is not the product of deliberate design. It emerged gradually as development practices changed, tools proliferated, and the scale of software grew beyond what any single team could manage. Each step forward in productivity created new dependencies and new trust assumptions. This chapter examines that trajectory, from monolithic codebases to AI-assisted development, and identifies the structural factors that make supply-chain attacks so attractive and so difficult to prevent.

The Monolith Era and Its Security Assumptions

Three decades ago, most applications were built as monoliths: single codebases written in-house, compiled with standard toolchains, and deployed to proprietary servers. A typical development team might write fifty thousand lines of code and depend on a half dozen libraries, most of which were either bundled with the operating system or obtained from commercial vendors.

This model had clear security properties. The organization controlled the code, the build environment, the deployment infrastructure, and the runtime. An attacker who wanted to compromise the application had to target the organization directly: break into a developer workstation, exploit a vulnerability in the application code, or gain access to the deployment server. The attack surface was limited and the trust boundaries were explicit.

The monolith model also had clear limitations. Development was slow, scaling was difficult, and each team had to reimplement common functionality. Organizations that needed more than one application ended up with duplicated code and duplicated effort. The economics of software development were pushing toward reuse and specialization, even before the internet made it practical.

The first crack in the monolith trust model appeared with component-based development. Frameworks like MFC, SWT, and early web frameworks

allowed developers to reuse code across projects. Shared libraries distributed with operating systems meant that applications no longer had to implement every function from scratch. These changes increased productivity but also introduced dependencies on code maintained by others. The shift was incremental and the risk was manageable because the ecosystem was small and largely controlled by known vendors.

Open Source as an Engine of Growth

The modern software supply chain would not exist without open source software. The GNU project, the Linux kernel, Apache, and later the explosion of open-source frameworks and libraries created an ecosystem where developers could build applications almost entirely from code written by others.

The economic driver was straightforward. If a functionality was needed by many organizations, and someone made it available under an open-source license, every other organization could use it instead of building it in-house. This reduced development cost and accelerated innovation. Organizations focused on their core business logic while relying on the open-source ecosystem for infrastructure, frameworks, tooling, and utilities.

By the early twenty-first century, open source was no longer an alternative but the foundation. Linux became the dominant operating system for servers. Apache and later Nginx served the majority of web traffic. Java, Python, and JavaScript frameworks enabled rapid development of complex applications. The number of available packages grew exponentially. The npm registry launched in 2010 with a few thousand packages. By 2024, it contained over two million packages and handled 4.5 trillion requests in a single year [8].

This growth created a fundamental change in the security model. An organization no longer controlled the code running in its applications. It depended on code maintained by volunteers and third-party organizations, built in environments it could not inspect, and distributed through registries it did not own. Trust was implicit: developers assumed that popular packages were safe because many people used them and any obvious problems would be noticed.

This assumption is reasonable for individual vulnerabilities in package code. Many eyes on code does catch problems, and the security communities around major projects are vigilant. But it does not protect against supply-chain

attacks, which do not rely on vulnerable code. They rely on compromising the process by which code is published, built, or distributed. A package can be perfectly secure in its source code and still become malicious if the maintainer is compromised, the build environment is tampered with, or the registry serves a modified version.

The CI/CD Revolution

Continuous integration and continuous deployment transformed how software is built and delivered, and they created a new layer of supply-chain complexity. Before CI/CD, builds were often manual or semi-automated processes executed on developer workstations or dedicated build servers. Someone ran a build script, checked the output, and manually deployed the result.

CI/CD automated this process. Code changes trigger automated workflows that run tests, build artifacts, perform security scans, and deploy to environments. Tools like Jenkins, Travis CI, CircleCI, and later GitHub Actions and GitLab CI became standard infrastructure. The benefits were substantial: faster feedback for developers, more consistent builds, fewer manual errors, and the ability to deploy changes continuously rather than in large, infrequent releases.

The automation introduced new trust dependencies. A CI/CD pipeline depends on the CI platform itself, the workflow definitions (which are code), the runners that execute jobs, the build tools and dependencies fetched during execution, and the credentials used to access external services. Each dependency is a potential attack surface.

The Codecov breach demonstrated how a supply-chain attack against CI/CD can scale. In early 2021, an attacker compromised the Bash Uploader script that Codecov customers ran in their CI pipelines, injecting code that exfiltrated secrets from the environment. The attack went undetected for two months and affected thousands of organizations, including HashiCorp, Twilio, and Confluent [9]. The attacker did not need to compromise any customer directly: they compromised a tool that customers trusted to run in their pipelines.

CI/CD also concentrated risk. A single pipeline often has elevated permissions: it can read secrets, push artifacts to registries, deploy to production, and interact with cloud infrastructure. Compromising a pipeline configuration

or a pipeline runner can grant an attacker broad access to an organization's infrastructure.

Containers, Cloud, and Infrastructure Abstraction

Containers added another layer of abstraction and another layer of supply-chain risk. Before containers, applications were typically deployed as standalone binaries, WAR files, or virtual machines. Deployment was platform-specific and environment drift was common: the software worked on a developer's machine but failed in production because of configuration differences.

Docker and containerization solved the portability problem by packaging applications with their dependencies in lightweight, portable images. A container image that runs locally will run identically in a CI pipeline, a staging environment, or production. The consistency and efficiency of containers made them dominant: Kubernetes became the standard orchestration platform, and cloud-native architecture became the default approach for new applications.

Containers introduced a new supply chain. Building a container requires a base image, typically pulled from Docker Hub or another registry. That base image is built by someone else, using their own dependencies and build process. When a Dockerfile specifies FROM ubuntu:22.04, the builder is trusting Canonical, Docker Hub, and the transport layer to deliver exactly that image, unmodified.

Container images are also large and opaque. A typical application container may contain hundreds of layers and thousands of packages. Scanning them for vulnerabilities is computationally expensive and produces results that are hard to interpret: a vulnerability may be present in an image but not reachable from the running application. Moreover, images are mutable by default: a tag like my-app:latest can be overwritten at any time, silently replacing one image with another. Organizations that pin to tags rather than digests are vulnerable to image substitution attacks.

Cloud infrastructure extended the trust boundary further. Applications no longer ran on servers owned and managed by the organization. They ran on compute instances, managed services, and serverless platforms provided by third parties. The organization depended on the cloud provider to maintain the integrity of the infrastructure, isolate tenants, and protect credentials. Cloud platforms also introduced managed registries, managed CI/CD, and

managed Kubernetes, each adding another layer of dependency and another trust assumption.

The abstraction was valuable: organizations could focus on application logic rather than hardware, operating systems, and networking. But it shifted security risk upstream. A vulnerability in the hypervisor, a compromised image in a shared registry, or a misconfiguration in the cloud provider's controls could affect every customer relying on that infrastructure.

The Explosion of Package Ecosystems

The modern software supply chain is fundamentally a dependency problem. Every language ecosystem developed its own package manager and registry, each with its own trust model, access controls, and security features. Understanding these ecosystems is essential because attacks target their specific weaknesses.

npm dominates JavaScript and TypeScript development. It has the largest ecosystem with over two million packages, but it historically had weak security controls. Until recently, account takeovers were common because password resets could be performed by anyone who controlled the email domain, even if that domain had expired. Typosquatting was rampant because npm allowed anyone to register a package with a similar name to an existing popular package. These weaknesses have been partially addressed, but the ecosystem's size and the low barrier to publishing make it an attractive target.

PyPI serves the Python ecosystem, which has grown rapidly due to data science, machine learning, and infrastructure automation. Python packages are typically installed into the same environment where application code runs, giving malicious packages direct access to the host system. PyPI historically had no mandatory authentication for publishing, making it easy for attackers to register malicious packages.

Maven Central serves Java, the language of enterprise software. It has stricter publishing requirements than npm or PyPI, but the ecosystem is enormous: a typical Java application depends on one hundred fifty or more components [10]. The complexity of the dependency graph makes it difficult to track all transitive dependencies and their vulnerabilities.

Other ecosystems include RubyGems for Ruby, NuGet for .NET, Cargo for Rust, Composer for PHP, and many others. Each has its own history, its

own security model, and its own set of known vulnerabilities. Attackers have targeted all of them.

The common pattern across ecosystems is that publishing is easy and verification is hard. Anyone can create an account and publish a package. Once published, the package becomes part of the ecosystem: it is downloaded by developers, included in projects, and potentially depended on by other packages. If it is later found to be malicious, the damage has already been done. Unpublishing is rare because it can break downstream projects. Yanking a version marks it as discouraged but does not prevent installation, and many package mirrors do not respect yank status.

When Software Began Generating Software

The latest phase in the evolution of the software supply chain is the introduction of generative AI into development workflows. Tools like GitHub Copilot, Amazon CodeWhisperer, and Claude Code suggest code to developers based on large language models trained on vast corpora of source code. These tools have improved developer productivity: studies report that developers using Copilot complete tasks faster and with higher satisfaction [11].

But AI introduces supply-chain risks that did not exist before. The first is code quality: AI-generated code contains security vulnerabilities at measurable rates. A 2023 study found that approximately thirty percent of Copilot-generated code snippets contained security weaknesses, including SQL injection, cross-site scripting, and use of insufficiently random values [12]. These are not unknown vulnerability classes, but they are introduced by a non-human process that does not understand the security implications of its output.

The second risk is hallucinated dependencies. AI models generate plausible-sounding package names that do not exist in any registry. When developers trust the AI and install these hallucinated packages, they are installing whatever the attacker who registered that name put in it. This attack vector, called slopsquatting, combines AI's hallucination tendency with the low barrier to package publishing [13]. Research found that roughly one in five AI dependency recommendations pointed to non-existent packages, and attackers have begun registering these names with malicious payloads [14].

The third risk is autonomous agents. The next generation of AI tools can do more than suggest code: they can edit files, run commands, manage

dependencies, and interact with CI/CD systems. An agent that can run npm install, git push, or deploy to production introduces execution-layer risks. If the agent is manipulated through prompt injection or if it is given excessive permissions, it can perform actions at machine speed that would have required human approval before.

AI does not create these risks from nothing. It exploits existing weaknesses in package registries, build systems, and CI/CD pipelines. But it amplifies them by automating the exploitation process and reducing the friction that previously slowed attackers. The same tools that accelerate development for legitimate engineers accelerate compromise for attackers.

The Structural Factors

The evolution described above was driven by economic incentives that are unlikely to reverse. Open source saves money. CI/CD accelerates delivery. Containers enable scale. Cloud infrastructure reduces capital cost. AI increases developer productivity. Organizations that reject these tools to simplify their security posture will be uncompetitive.

The structural factors that make supply-chain attacks attractive are baked into this evolution:

- Concentration: A small number of platforms (GitHub, npm, PyPI, Docker Hub) serve a vast majority of the industry. Compromising one platform affects thousands of organizations.
- Opacity: Most developers cannot see or audit the hundreds of transitive dependencies in their projects. Trust is implicit rather than verified.
- Automation: CI/CD and deployment automation mean that a compromise propagates quickly from code to production without human review.
- Perimeter erosion: Software runs on infrastructure owned by third parties, in environments shared with other tenants, using credentials managed by external systems. There is no clear perimeter to defend.
- Asymmetric risk: Maintainers of popular open-source projects contribute voluntarily and bear disproportionate risk. They have no organizational backing but their software may run in banking systems, healthcare platforms, or government infrastructure.

None of these factors can be eliminated. The goal of supply-chain security is to manage them: to reduce opacity through visibility, to replace implicit trust with cryptographic verification, to slow the propagation of compromise through controls and checks, and to distribute risk so that a single failure does not have catastrophic consequences.

Understanding how we arrived here clarifies why supply-chain security is difficult and why there is no simple solution. It also clarifies what is possible: the same forces that created complexity (automation, standardization, cryptographic infrastructure) can be harnessed to create defense. The next chapter examines the principles that guide that defense.

Chapter 3: Foundational Security Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Zero Trust and Its Real Meaning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Least Privilege at Every Boundary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Defense in Depth Without Illusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Assume Breach, Verify Everything

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Separation of Duties in Automated Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Security as Engineering, Not Policy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 4: Source Control and Code Ownership

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Git and the Trust Model of Version Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Repository Access and Identity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Code Review as a Security Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Branch Protection and Release Controls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Fork-Based Development and External Contributors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Secrets in Source and Accidental Exposure

Chapter 5: Dependency Ecosystems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

How Package Managers Actually Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Growth of Transitive Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Trust Models for Package Publishers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Version Resolution and Its Security Implications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Lockfiles, Pinning, and Reproducibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Private vs. Public Registries and Proxy Caching

Chapter 6: Build Systems and Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Build as a Transformation Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Build Dependencies and Their Risks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Reproducible and Deterministic Builds

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Build Environment Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Caching and Its Attack Surfaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Artifact Integrity from Source to Output

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 7: CI/CD Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What CI/CD Automates and What It Trusts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Pipeline Definitions as Code and as Targets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Runner Security and Shared Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Secrets in Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Cross-Repository and Trigger Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Designing for Integrity and Observability

Chapter 8: Containers and Image Supply Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Container Image as a Supply Chain Artifact

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Base Image Trust and Inherited Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Image Building and Multi-Stage Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Registry Security and Access Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Scanning Images and Knowing What to Do With Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Signing Images and Enforcing Admission Policies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 9: Infrastructure as Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

IaC as Code and as Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Module and Provider Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

State Management and Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Drift Detection and Unauthorized Changes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Policy as Code and Guardrails

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Supply Chain of Platform Definitions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 10: Deployment and Runtime Environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Trust Chain Into Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Environment Isolation and Segmentation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Deployment Strategies and Rollback Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Runtime Protection and Attack Surface Reduction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Service Meshes and Zero Trust Networking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Observability as a Security Signal

Chapter 11: Package Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Typosquatting and Lookalike Packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Dependency Confusion and Internal Name Shadowing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Malicious Packages and Dropper Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Compromised Maintainers and Account Takeover

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Dependency Substitution and Version Manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 12: Repository and Code Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Poisoned Commits in Third-Party Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Malicious Pull Requests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Repository Compromise and Key Signing Key Theft

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Maintainer and Organization Takeover

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Social Engineering Against Open Source

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Case Study: XZ Utils and the Anatomy of a Near Miss

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 13: Build and Pipeline Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Build Tool and Plugin Compromise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

CI/CD Configuration Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Runner Exploitation and Shared Infrastructure Abuse

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Artifact Tampering Between Build and Deploy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Supply-Chain Injection via Environment Variables

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Case Study: SolarWinds and the Build-Environment Attack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Case Study: Codecov and the CI Script Compromise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 14: Container and Infrastructure Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Poisoned Container Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Registry Compromise and Access Escalation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Malicious IaC Modules and Providers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Infrastructure Hijacking via Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Kubernetes Supply Chain Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 15: Secrets, Keys, and Identity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Where Secrets Live and How They Leak

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Hard-Coded and Accidental Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Key Management for Signing and Encryption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Identity in Automated Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Rotating Credentials and Revoking Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 16: Provenance and Attestation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What Provenance Means and Why It Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Provenance Formats and Specifications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Generating Attestations in Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Verifying Attestations Downstream

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Limits of Provenance and Trust on First Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 17: SBOMs and Visibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What an SBOM Is and What It Is Not

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

SPDX, CycloneDX, and Other Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Generating SBOMs Across Build Stages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Using SBOMs for Vulnerability Assessment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

SBOM Accuracy and False Positives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

SBOMs in Regulation and Compliance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 18: Digital Signatures and Trust Infrastructures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What Signatures Protect and What They Do Not

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Key Management and Signing Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Sigstore, Fulcio, and Rekor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Package Signing and Trusted Publishing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Signing Keys as High-Value Targets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Building a Signing Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 19: SLSA and Security Frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What SLSA Specifies and Why It Exists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Levels and What They Require

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Achieving SLSA Compliance Incrementally

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

SLSA and Other Standards: NIST, CISA, OpenSSF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

When Frameworks Help and When They Hinder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Measuring Security Posture Without Theater

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 20: Securing AI-Assisted Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

How AI Coding Assistants Actually Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Hallucinated Dependencies and Nonexistent Packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Vulnerability Patterns in AI-Generated Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Prompt Injection in Development Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

AI Tools as Supply-Chain Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Securing the AI Development Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 21: Autonomous Agents and the Next Frontier

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What Autonomous Agents Can Do Today

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Agent Identity and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Terminal and Environment Access

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Repository Interaction and Merge Authority

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Production Deployment by Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Designing Safe Agent Environments

Chapter 22: AI Models, Plugins, and Datasets as Dependencies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Models as Software Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Dataset Provenance and Poisoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Plugin and Extension Trust

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Fine-Tuning and Custom Model Supply Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Registry Security for AI Artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 23: Reference Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Small Team: Maximum Security with Minimum Overhead

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Growing SaaS Company: Scaling Controls with Velocity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Large Enterprise: Governance at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Regulated Organization: Compliance Without Paralysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Open Source Project: Security Without Central Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

The Cloud-Native Platform: Supply Chain as a Service

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 24: Assessing and Hardening Your Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Creating an Inventory of Your Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Mapping Trust Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Threat Modeling for Supply Chains

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Prioritizing Risks and Quick Wins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Establishing Security Policies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Rolling Out Controls Incrementally

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 25: Operations, Monitoring, and Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What to Monitor in the Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Detection Patterns and Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Incident Response for Supply-Chain Compromise

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Recovery, Reroll, and Remediation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Post-Incident Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Metrics That Matter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Chapter 26: Governance and Organization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Who Owns the Supply Chain

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Responsibilities Across Teams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Vulnerability Disclosure and Coordination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Patching, Exceptions, and Risk Acceptance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Third-Party and Vendor Risk

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Reporting Up and Measuring Progress

Chapter 27: Tradeoffs and Engineering Decisions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Security vs. Velocity: Beyond the False Dichotomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Automation vs. Human Judgment

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Centralized vs. Distributed Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Open Source Contributions and Security Friction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

When Controls Create False Confidence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Making Tradeoffs Explicit and Sustainable

Chapter 28: The Future of Software Supply-Chain Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Increasingly Autonomous Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Code Review in the Age of AI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

New Models of Trust and Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Continuous Generation and Continuous Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

Regulatory Trajectories

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

What Will Last and What Will Change

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thesoftwaretrustchain.