Jit

Connect with Jit

Application Security > Security

STRIDE Threat Model: A Complete Guide

By **Charlie Klein**

Published April 28, 2025.

In this article

Show More

You can't protect what you don't know exists. With attack surfaces expanding across microservices, third-party APIs, and complex cloud architectures, predicting where the subsequent breach might come from is more complicated than ever.

Cybersecurity budgets grew by 8% in 2024. On paper, that sounds like progress. But consider this: phishing attacks have surged by over 1,265%, with credential phishing increasing by 967% since late 2022. Throwing more money at the problem clearly can't be the only solution.

The STRIDE threat model offers a structured approach to identifying vulnerabilities, staying ahead of potential threats, and integrating security into modern workflows — all without disrupting your development pipeline.

What is the STRIDE Threat Model?

The STRIDE Threat Model is a structured approach to identifying and categorizing security risks in software systems. Originally developed by Microsoft to help teams systematically analyze potential threats, this model breaks threats down into six categories:

1. Spoofing
2. Tampering
3. Repudiation
4. Information Disclosure
5. Denial of Service (DoS)
6. Elevation of Privilege

As API sprawl and third-party integrations expand your attack surface daily, STRIDE must evolve into a dynamic, continuously validated process. This continuous iteration is especially crucial in cloud environments, where the volume and flow of cloud computing data introduce new security risks, particularly around Information Disclosure and Tampering. Without embedding STRIDE directly into CI/CD workflows and runtime environments, most teams are left with outdated threat models that can't keep up with production.

Breaking Down STRIDE: Understanding Each Threat Category

1. Spoofing

Spoofing is when an **attacker impersonates a trusted service or user by falsifying their identity.** Attackers often use compromised API tokens, machine identities, or federated credentials to gain unauthorized access to systems and data.

Attackers **exploit weaknesses in managing credentials,** such as overly broad token permissions or insecure storage practices. For example, long-lived service tokens or API keys improperly stored in source code or CI/CD pipelines can be an easy entry point. **Once inside, attackers escalate privileges** by taking advantage of weak identity management policies and

overly permissive access controls. They can then **move laterally across systems and access sensitive resources.**

How to mitigate this threat:

- Enforce **MFA** (hardware keys or app-based TOTP).
- Use **short-lived, scoped OAuth tokens** via Okta or Auth0.
- Automate **secrets rotation** with AWS Secrets Manager or HashiCorp Vault.
- Continuously scan for hardcoded secrets using **Semgrep**.

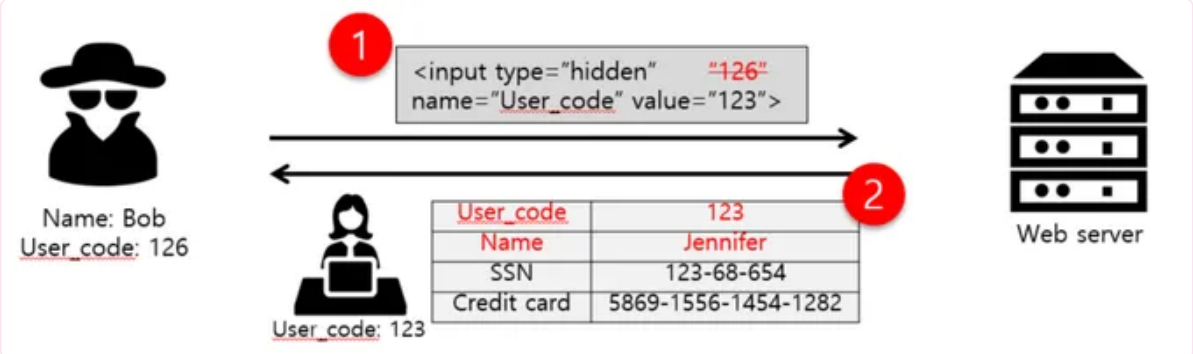
2. Tampering

Tampering involves **unauthorized modification of code, configuration files, or data.** It often happens through compromised CI/CD pipelines, manipulated IaC templates, or malicious code injections during the software build.

Attackers use tampering methods to **alter the behavior of applications or infrastructure** and introduce vulnerabilities, bypass security controls, or manipulate functionality. Whether it's **stealing data, disrupting service operations, or gaining control over systems,** tampering undermines the integrity of the entire software supply chain and poses significant security risks.

How to mitigate this threat:

- Sign all build artifacts using **Sigstore or Cosign.**
- Enforce the **least privileged access** across CI/CD systems.
- Automate integrity checks with **hash validation** at each deployment stage.



3. Repudiation

Repudiation happens when **users deny performing specific actions, and the system lacks verifiable logs to prove otherwise.** The absence of tamper-evident audit trails creates blind spots that make incident response and forensic investigations nearly impossible.

This isn't just a logging issue; it's a systemic failure when actions aren't traceable and verifiable. According to IBM, **the average time to identify and contain breaches without centralized logging and audit trails is 58% longer,** leading to significantly higher regulatory penalties and breach costs. Without append-only, integrity-validated logs, attackers can erase their tracks, prolong dwell time, and escalate the impact—turning what should be a contained incident into a systemic failure.

How to mitigate this threat:

- Implement **append-only, tamper-evident logs** (e.g., AWS CloudTrail with integrity validation).
- Centralize log collection and analysis via **SIEM tools** like Splunk or Datadog.
- Digitally sign logs at the point of creation.

4. Information Disclosure

Information disclosure is one of the most avoidable yet consistently mishandled risks. It happens when **sensitive data, such as customer records, credentials, or proprietary code, is exposed to unauthorized parties.** This often stems from misconfigured cloud storage, overly permissive API responses, or careless access control policies.

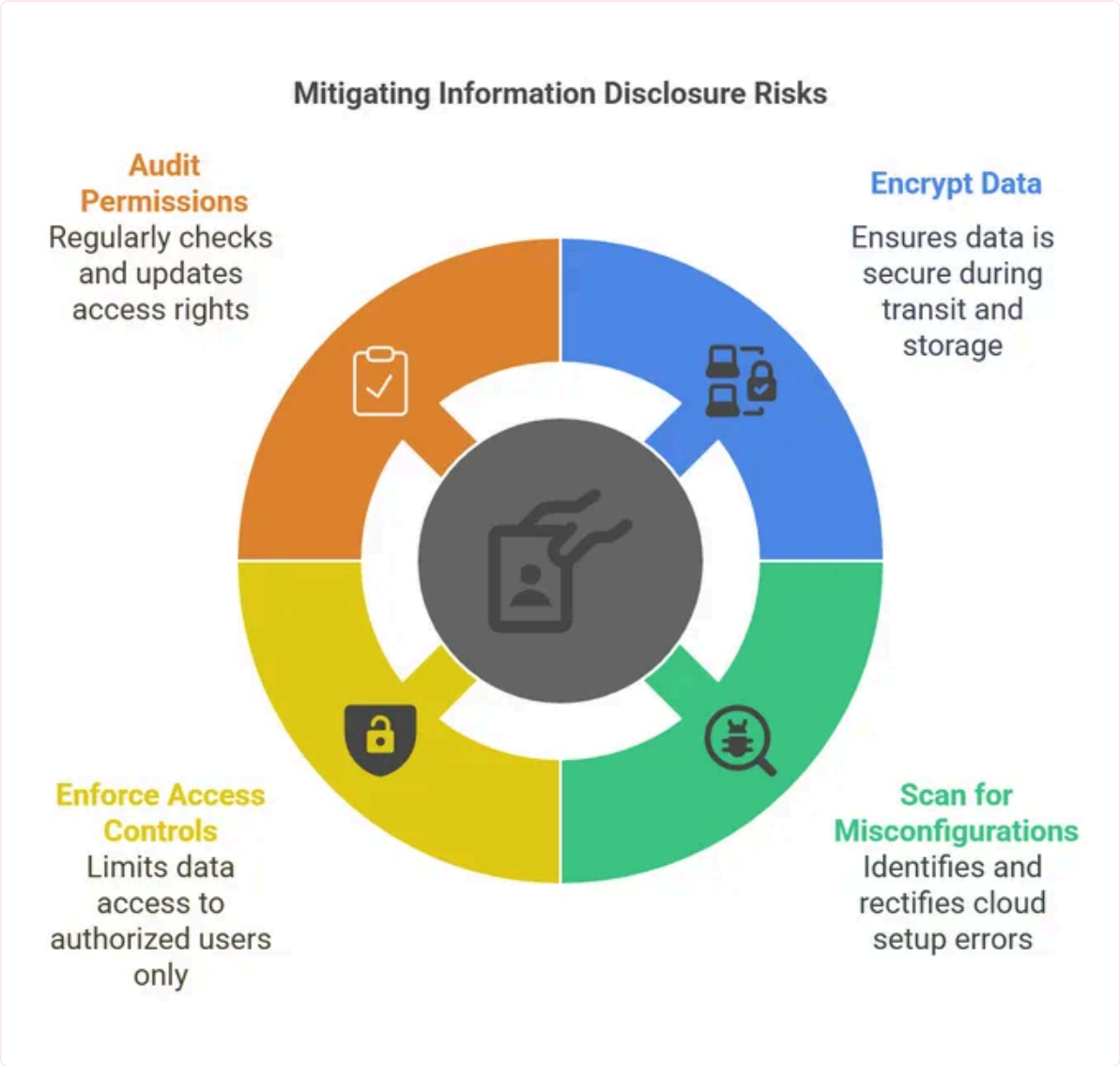
Prioritizing speed over security often leads to sloppy configurations like publicly exposed S3 buckets or unprotected Kubernetes secrets. In rapidly scaling environments, visibility over where data resides and who can access it is frequently lost.

How to mitigate this threat:

- **Encrypt data** in transit (TLS 1.2+) and at rest (AES-256).
- Continuously **scan for cloud misconfigurations** with tools like Trivy or KICS.

Review and enforce **strict access controls**, limiting API responses to only the necessary data.

Regularly **audit permissions** across all cloud storage and services.



5. Denial of Service (DoS)

Denial of Service (DoS) attacks **target exposed APIs, load balancers, and application endpoints by flooding them with excessive traffic.** Attackers exhaust compute, memory, and network bandwidth allocations, leaving systems vulnerable.

Moderate volumetric or protocol-based attack traffic can cause severe disruptions in elastic multi-tenant environments. It **saturates ingress points, drains connection pools, and triggers uncontrolled auto-scaling events.** As a result, performance degrades, latency rises, and cascading failures affect dependent microservices.

The impact is significant: **35% of organizations targeted by DDoS attacks experience service disruptions,** with 14% reporting significant financial losses due to unmitigated scaling and resource exhaustion.

How to mitigate this threat:

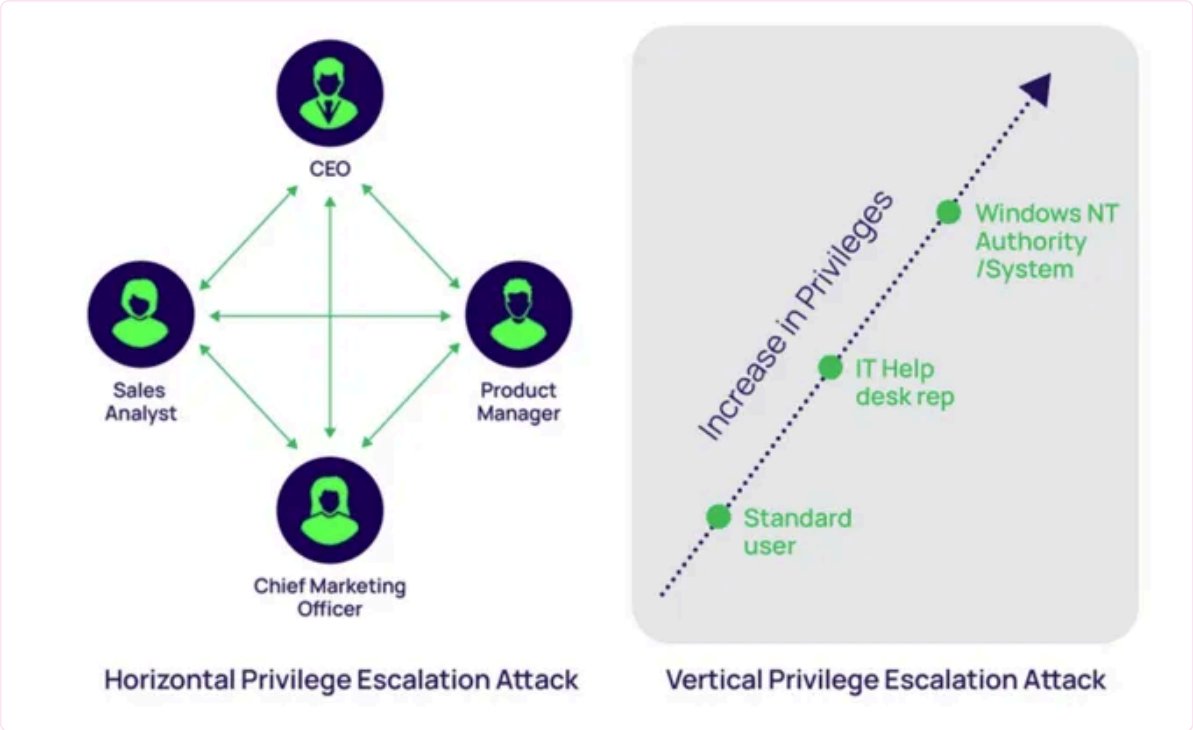
- Apply **strict rate limiting and request throttling** on all public APIs.
- **Implement circuit breakers and timeouts** to prevent cascading failures across services.
- Use **managed DDoS protection** like AWS Shield Advanced or Cloudflare.
- **Continuously validate resource configurations** to avoid open-ended scaling policies.

6. Elevation of Privilege

Elevation of Privilege (EoP) happens when **an attacker gains access to higher permissions than initially granted.** This often stems from over-permissioned IAM roles, weak role-based access controls (RBAC), or compromised service accounts that enable lateral movement and escalation in modern applications. Dev teams often take shortcuts with permissions—granting broad access to speed up development. These oversights create easy privilege escalation paths.

How to mitigate this threat:

- Enforce **strict least privilege policies,** granting users and services the minimum access they need.
- Implement **just-in-time (JIT) access controls** for sensitive actions with automatic expiration.
- Regularly **audit IAM roles,** removing excessive permissions and revoking unused access.
- **Monitor for abnormal privilege escalation attempts through your SIEM** or cloud monitoring tools.



When to Apply the STRIDE Threat Model

Apply STRIDE during architectural design before deploying microservices, exposing APIs, or onboarding third-party vendors. Skipping threat modeling at this stage introduces undocumented trust assumptions and unmitigated attack paths.

But STRIDE isn't a one-time exercise. When you migrate to multi-cloud, refactor core components, or adjust IAM policies, you must reassess this model. These are critical points where trust boundaries shift and new attack vectors open up.

For instance, if you implement or modify AWS IAM Identity Center, it's crucial to model how identity management and access controls are handled across your environment. Misconfigurations or overly permissive roles can expose sensitive data or grant unnecessary access, especially as you onboard new applications or third-party integrations.

STRIDE should also be part of compliance efforts, as frameworks like SOC 2, ISO 27001, and HIPAA often require formal threat modeling. Automating STRIDE processes ensures your models stay current and support compliance.

How to Implement STRIDE in Your Security Strategy

1. Embed STRIDE Threat Modeling in Architectural Reviews with Mapped Controls

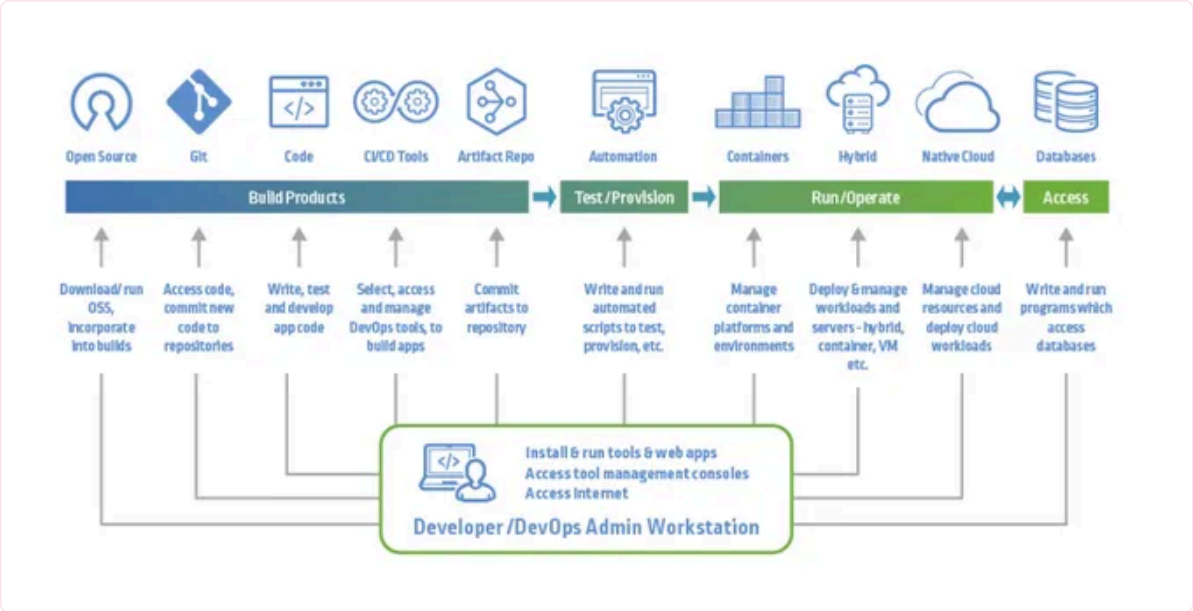
STRIDE threat modeling should be a **required step in architectural design reviews.** It should focus on **defining trust boundaries, data classifications, and privilege domains** for every new microservice, API, and third-party integration.

Prioritize components handling identity federation, token lifecycles, and session management, where the risks of spoofing and privilege escalation are highest. For each identified threat, **map enforceable security controls.** For example, spoofing requires short-lived, scoped tokens and MFA; tampering demands signed builds and artifact integrity validation.

2. Automate Security Control Validation in CI/CD Pipelines

Automate security control validation by **integrating threat model-driven policies directly into your CI/CD pipelines.** Run tools like Semgrep for SAST, Trivy for container scanning, and OSV-scanner for dependency checks automatically on every build, blocking deployments that violate predefined STRIDE-mapped controls.

Enforce requirements like signed commits, validated IaC configurations, and mandatory integrity checks on all build artifacts. This enforcement guarantees that security policies are continuously applied, version-controlled, and automated as part of the software delivery lifecycle.



3. Apply Continuous Policy Enforcement Across Runtime Environments

Extend enforcement beyond CI/CD **by embedding policy-as-code frameworks like Open Policy Agent (OPA)** directly into your service mesh and API gateways. Enforce **granular authorization, apply rate limits, and continuously validate infrastructure state** against your threat model. When drift or violations are detected, trigger automated remediation or rollback to maintain policy compliance and prevent privilege escalation in live environments.

4. Operationalize Threat Model Reviews Based on System Change and Threat Intelligence

Integrate threat model reviews into your continuous engineering and security workflows, not as periodic compliance tasks but as a core component of operational risk management. **Enforce STRIDE reassessment whenever your team onboards new third-party vendors** or modifies authentication and authorization mechanisms.

Enrich these reviews with real-time threat intelligence, incident root cause analyses, and vulnerability disclosures from bug bounty programs to drive iterative improvements.

Ensure threat model updates are **tightly version-controlled, with changes seamlessly integrated into architecture documentation, security control libraries, and CI/CD validation gates.**

Future-Proofing Threat Modeling with Jit

As your attack surface grows, threat modeling can't be a one-time exercise or a static artifact. Your team needs continuous, automated threat models embedded in development and runtime environments.

Jit's Product Security platform operationalizes STRIDE by turning your threat model into an enforceable, living framework. It orchestrates tools like Semgrep, Trivy, and OSV-scanner directly within CI/CD pipelines, ensuring every commit and deployment is evaluated against pre-defined STRIDE-mapped policies.

Jit's policy-as-code makes security controls versionable, auditable, and enforceable across services and environments, eliminating configuration drift and ensuring real-time adaptation. By integrating directly with developer workflows, Jit helps teams identify, prioritize, and remediate risks without slowing down delivery. **Learn more here.**

Related Articles

A Step-by-step Guide to Preventing Dependency Confusion Attacks

Raz Probstein March 17, 2025

10 Malicious Code Examples You Need to Recognize to Defend Your SDLC

Dotan Agmon June 12, 2024

Secure Coding: Top R… Practices, and Techniques

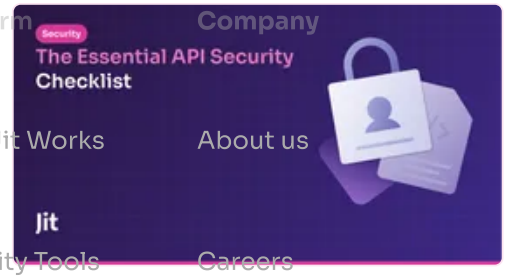
Charlie Klein February 7, 2025

Platform
How Jit Works
Security Tools
Integrations
Jit vs. Snyk
User Stories

Company
About us
Careers
Events
News

Resources
Blog
Docs
Pricing
Customers
Platform Security

Contact us
Book a Demo
Start Free



The Essential API Security Checklist

Charlie Klein April 8, 2025

What is CVE-2024-44308 (XSS), and How to Protect From It

Charlie Klein May 2, 2025

100 Summer Street
Boston, MA, 02110
USA