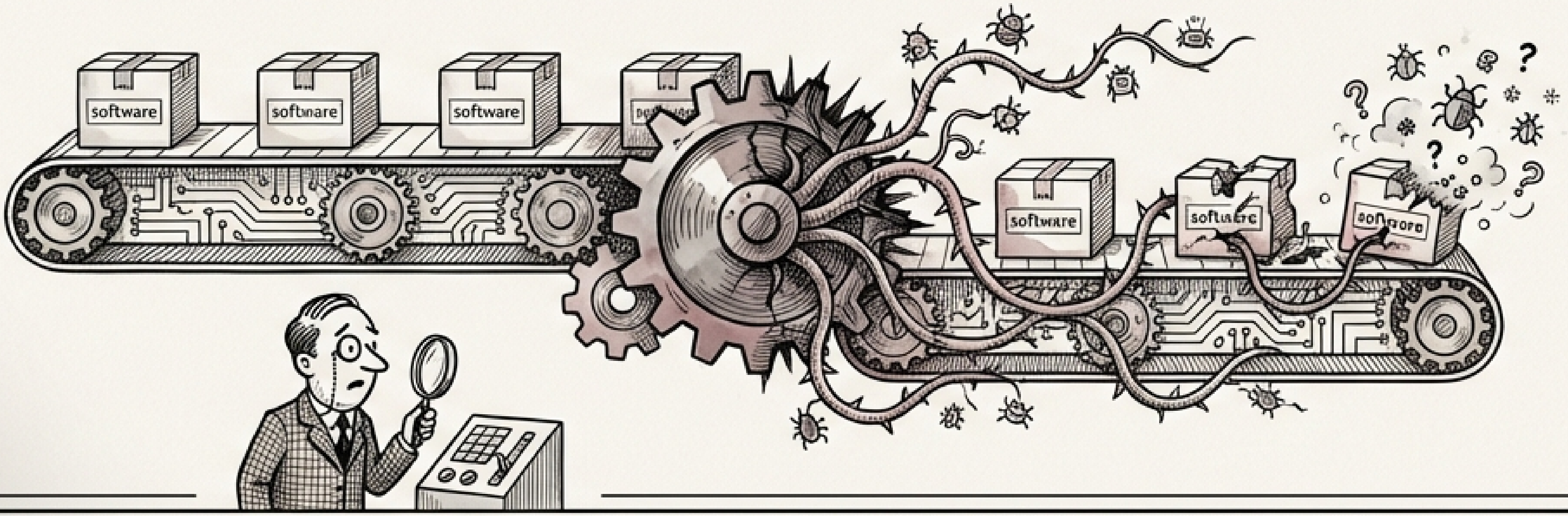
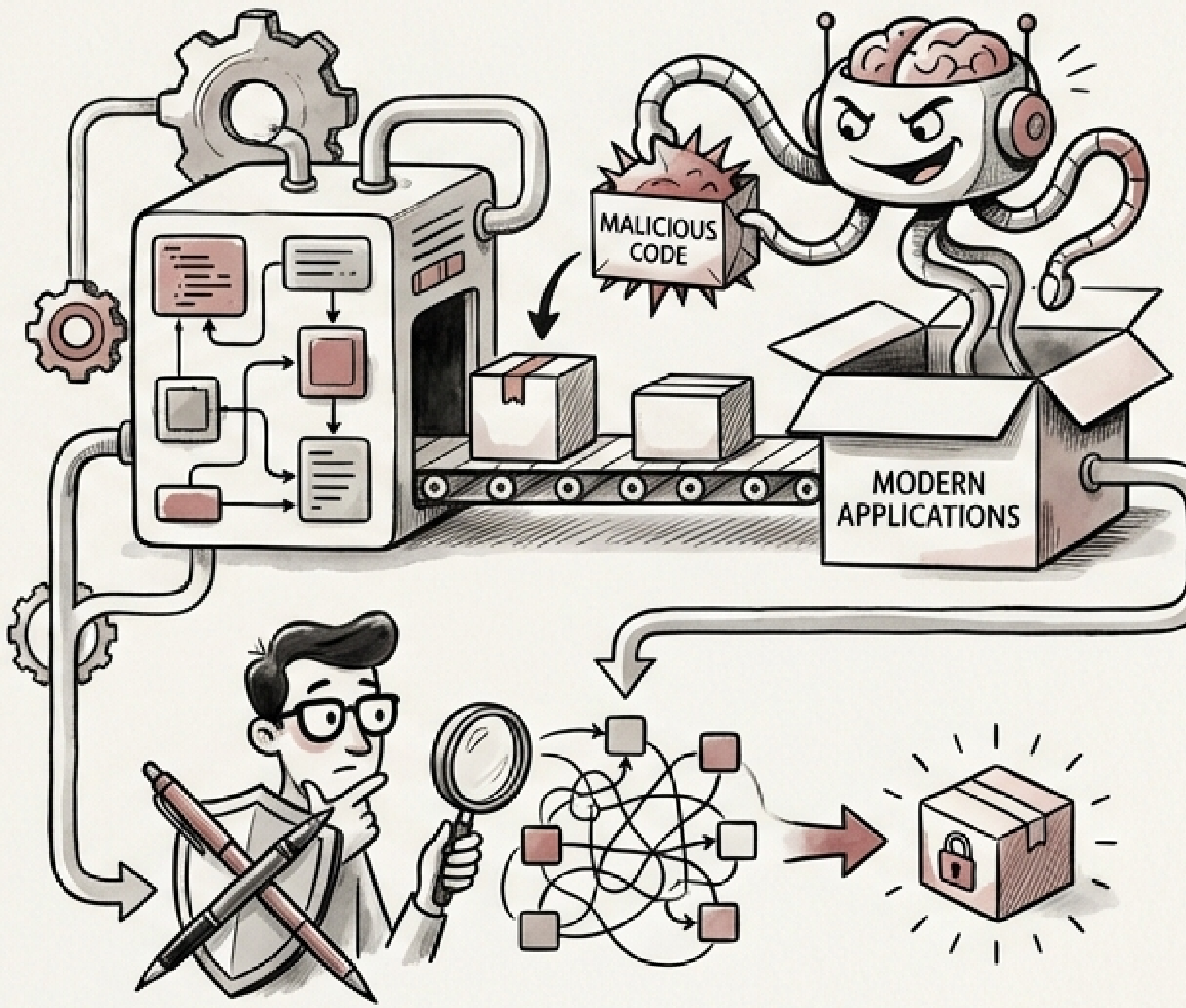
A Present Day Dilemma

The Growing Threat: Software Supply Chain Attacks



THE GROWING THREAT: SOFTWARE SUPPLY CHAIN ATTACKS

- Modern applications rely heavily on third-party code, with approximately 85% of the codebase often sourced externally.
- Supply chain attacks represent the fastest-growing threat vector in cybersecurity, demanding immediate attention.
- AI-augmented development amplifies supply chain risks due to AI tools suggesting dependencies and generating imports.
- AI tools can inadvertently introduce malicious or non-existent packages into the software development lifecycle, increasing vulnerability.
- Developers must understand and mitigate these risks to maintain secure and reliable applications.



Recent High-Profile Supply Chain Breaches

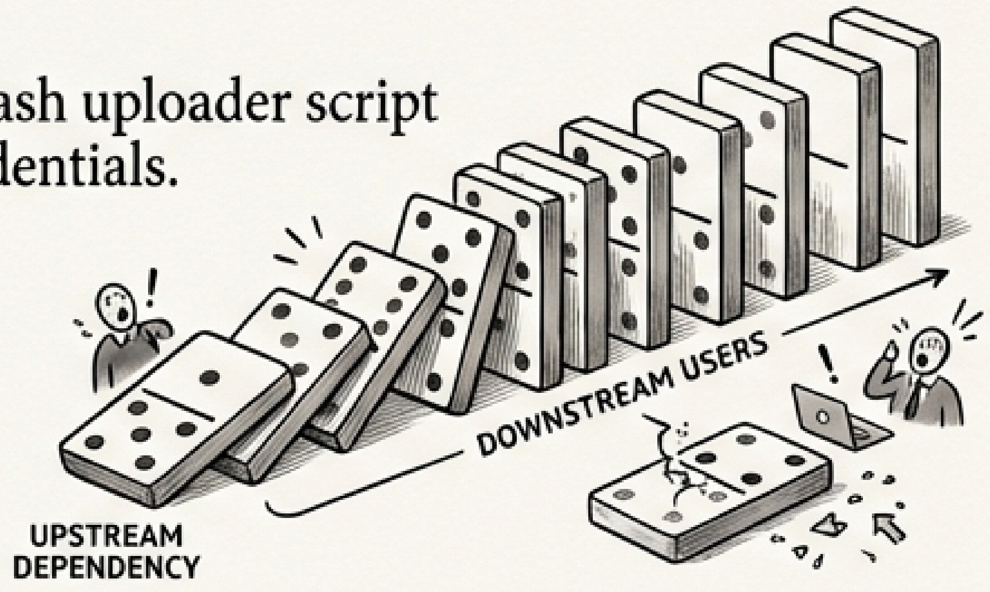
TIMELINE OF NOTABLE INCIDENTS

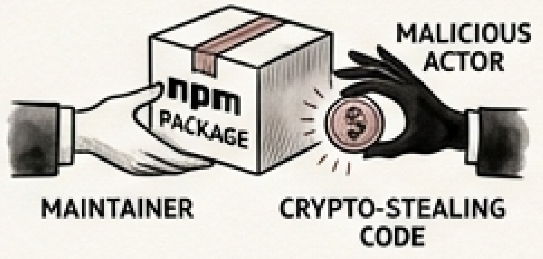
A nation-state actor compromised the build system, affecting over 18,000 organizations.

Attackers modified the bash uploader script to exfiltrate sensitive credentials.

Social engineering was used over two years to insert a backdoor into the widely used compression tool.

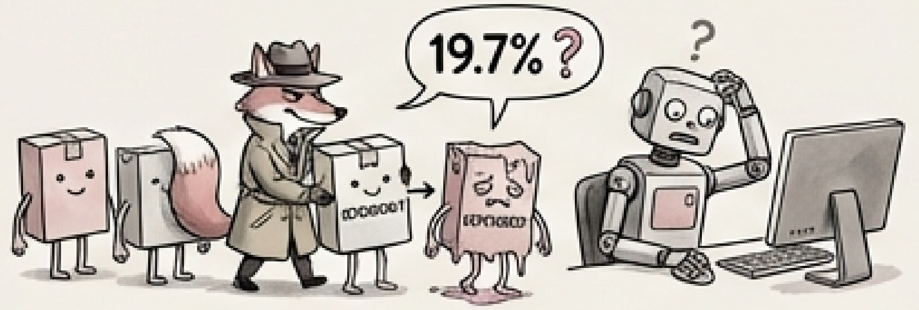
- A nation-state actor compromised the build system, affecting over 18,000 organizations.
- Attackers modified the bash uploader script to exfiltrate sensitive credentials.
- xz Utils (2024)
  A cascading supply chain attack occurred through.
- 3CX (2023)
  A cascading supply chain attack occurred through a compromised upstream dependency, impacting numerous downstream users.
- Event-stream (2018)
  A maintainer handoff led to a malicious actor adding crypto-stealing code to the popular npm package.



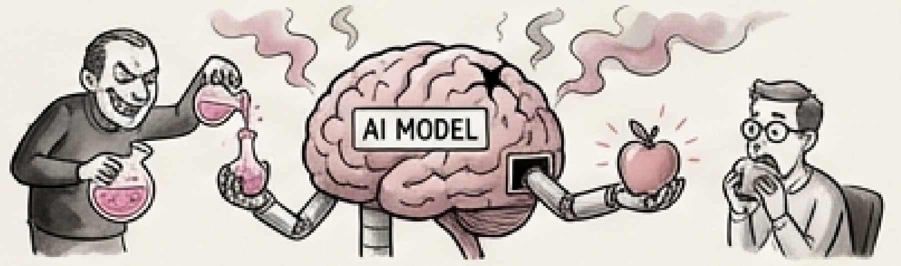


AI-Specific Risks: A New Breed of Supply Chain Attacks

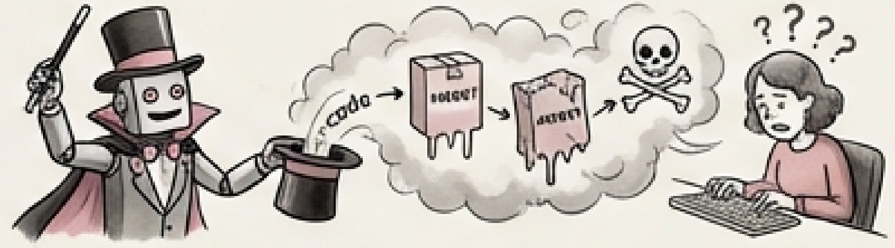
- **Slopsquatting:** Attackers register packages that are similar to AI-recommended dependencies but do not exist. 19.7% of AI recommendations are for non-existent packages.



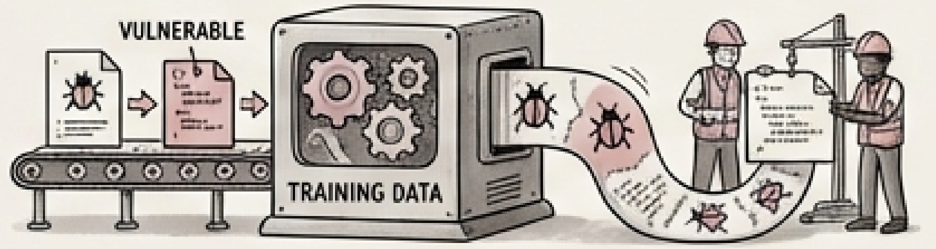
- **Model poisoning:** Compromised AI models can suggest vulnerable code patterns or backdoored dependencies to developers.



- **AI hallucinated imports:** AI may generate import statements for packages that seem plausible but are actually malicious or nonexistent.

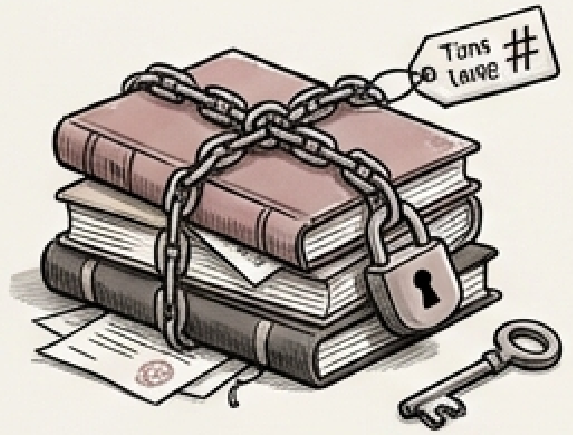


- **Training data contamination:** AI tools trained on code containing vulnerabilities may inadvertently reproduce those patterns in new code.



- **Plugin/MCP (Machine Code Patching) supply chain attacks:** Malicious AI tool extensions can exfiltrate code or inject vulnerabilities.

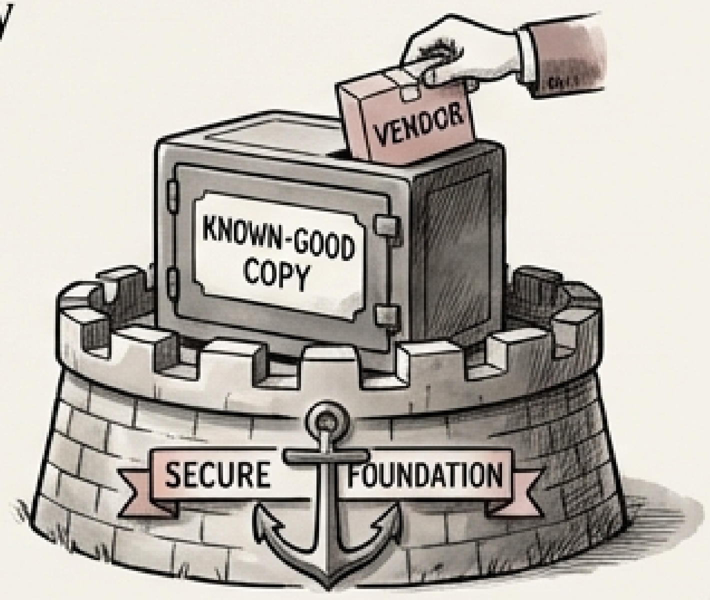
Dependency Management Fundamentals: Your First Line of Defense



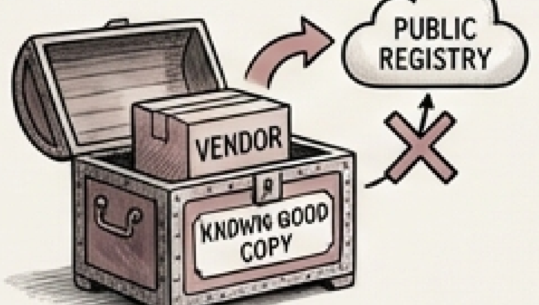
- **Lockfiles:** Pin exact versions of dependencies with integrity hashes (e.g., `package-lock.json`, `Pipfile.lock`, `go.sum`) to ensure consistent builds.

- **Dependency review:** Carefully evaluate new dependencies for their security posture, maintainer reputation, and community health.

- **Minimum dependency principle:** Reduce the attack surface by minimizing the number of dependencies in your project.

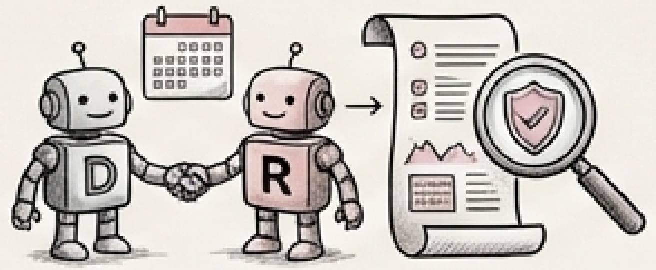


- **Vendoring:** For critical dependencies, vendor a known-good copy rather than pulling from a public registry at build time. This reduces reliance on external sources.

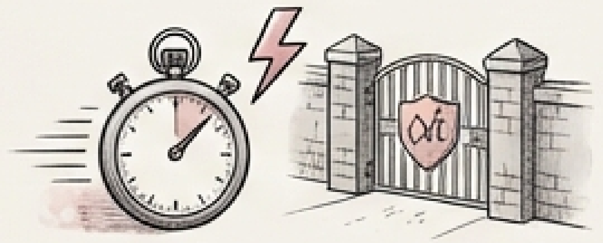


- These fundamentals create a strong foundation for secure dependency management.

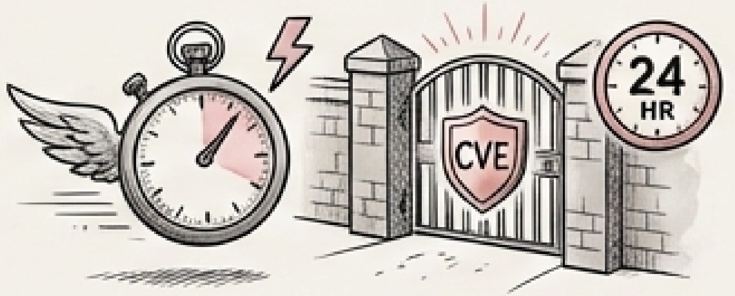
Automated Dependency Updates: Balancing Security and Stability

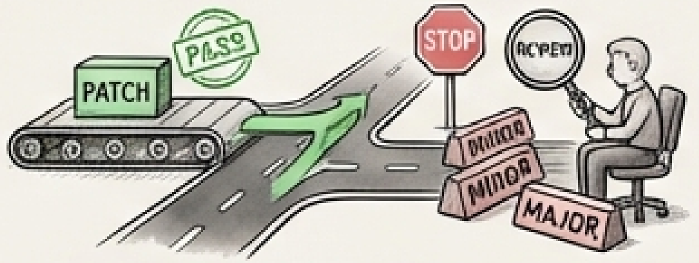


- **Dependabot/Renovate:** Automate the creation of pull requests for dependency updates, providing changelogs and vulnerability context.

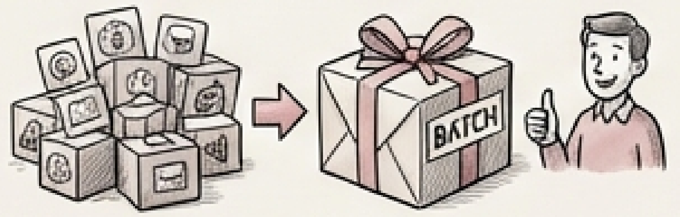


- **Security updates:** Implement an expedited path for critical CVEs, aiming to patch them within 24 hours of disclosure.

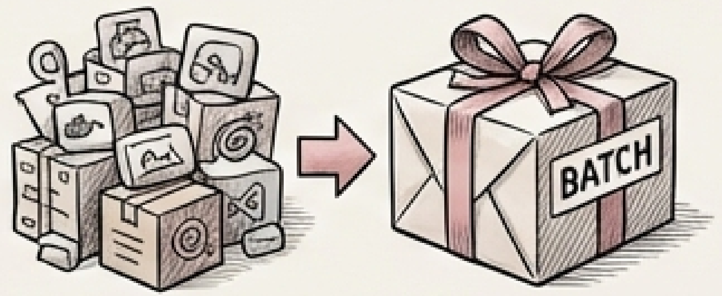




- **Update strategy:** Automatically merge patch versions with passing tests, while requiring manual review for minor and major versions.



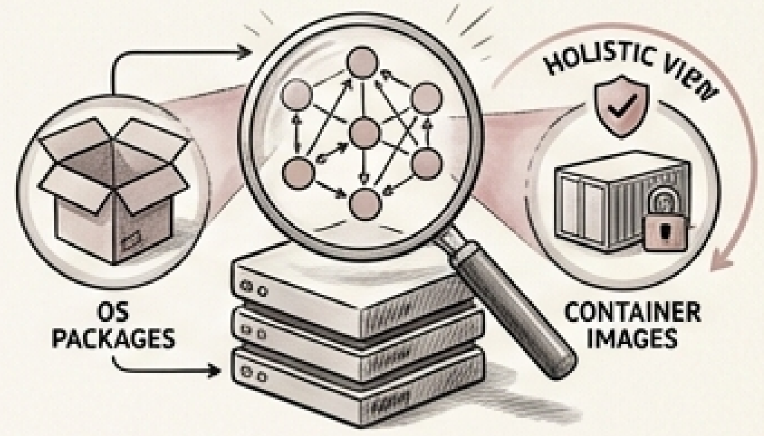
- **Grouped updates:** Batch related dependency updates to reduce pull request noise and streamline the review process.

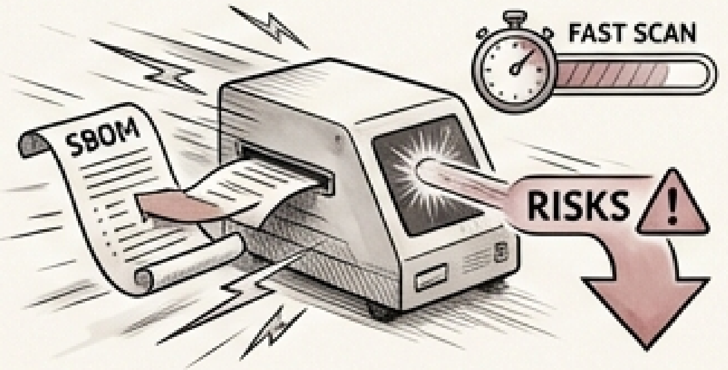


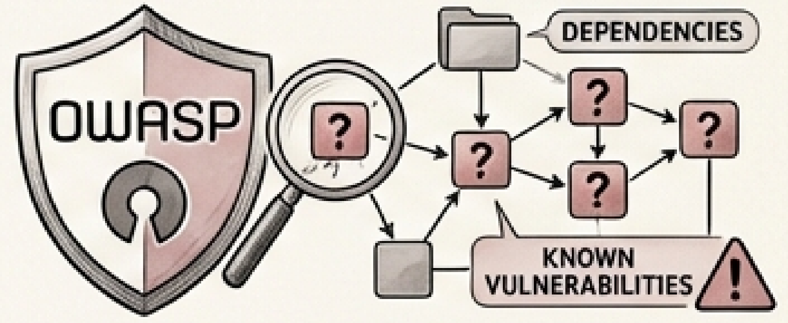
- Automated updates help maintain security and stability with less manual effort.

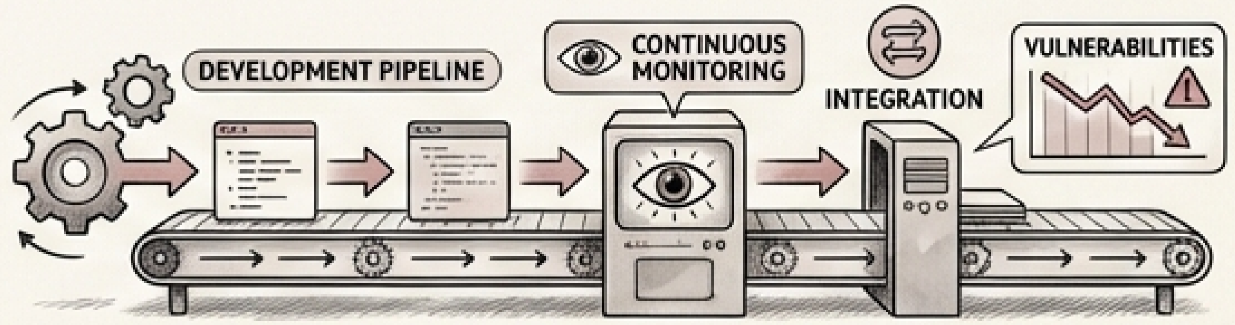
SCA Tools: Identifying and Remediating Vulnerabilities

- **Snyk:** Offers developer-friendly SCA with IDE integration, fix suggestions, and reachability analysis to prioritize vulnerabilities.
- **Trivy:** Provides comprehensive scanning covering OS packages, language dependencies, and container images for a holistic view of your security posture.
- **Grype:** Delivers fast vulnerability scanning based on Software Bill of Materials (SBOMs), enabling efficient identification of risks.
- **OWASP Dependency-Check:** A free, open-source SCA tool for identifying known vulnerabilities in project dependencies.
- SCA tools continuously monitor for vulnerabilities and can be integrated into the development pipeline.



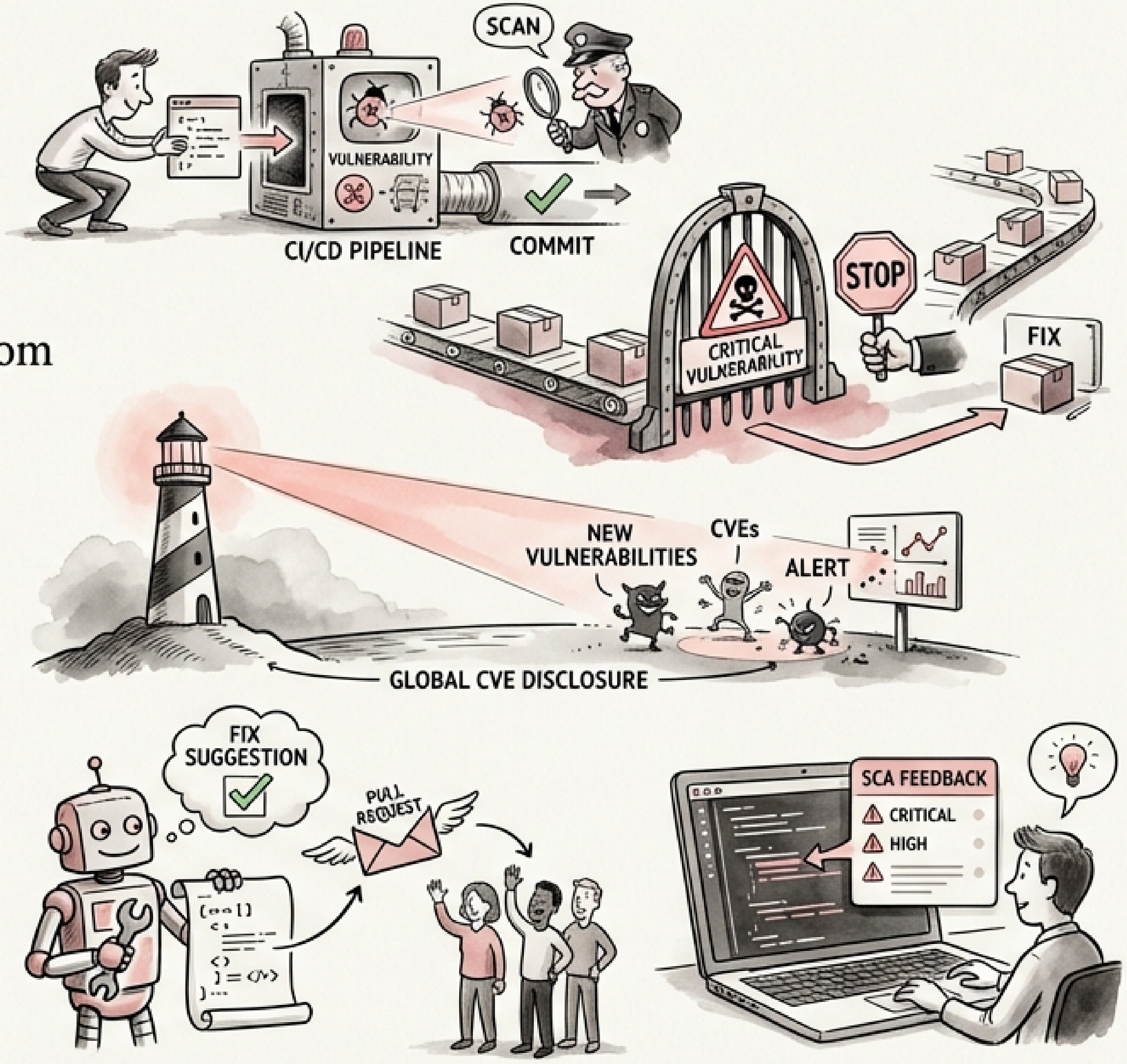




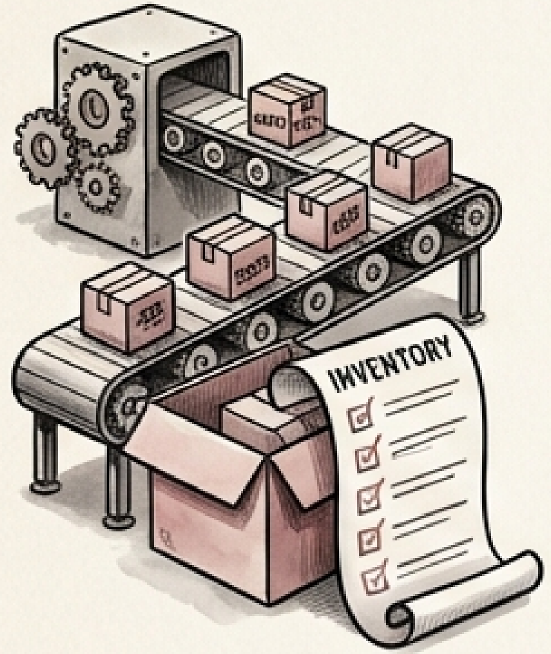


Integrating SCA into Your Workflow: Shift Security Left

- **Scan on every commit:** Integrate SCA tools into your CI/CD pipeline to scan for vulnerabilities with each commit.
- **Block builds with critical vulnerabilities:** Configure your CI/CD pipeline to prevent builds from proceeding if critical vulnerabilities are detected.
- **Continuous monitoring for new CVEs:** Implement continuous monitoring to detect new vulnerabilities as they are disclosed.
- **Automated fixes:** Use SCA tools to generate automated fix suggestions and pull requests for identified vulnerabilities.
- **Provide developer feedback:** Integrate SCA results directly into developer IDEs to provide immediate feedback on vulnerabilities.

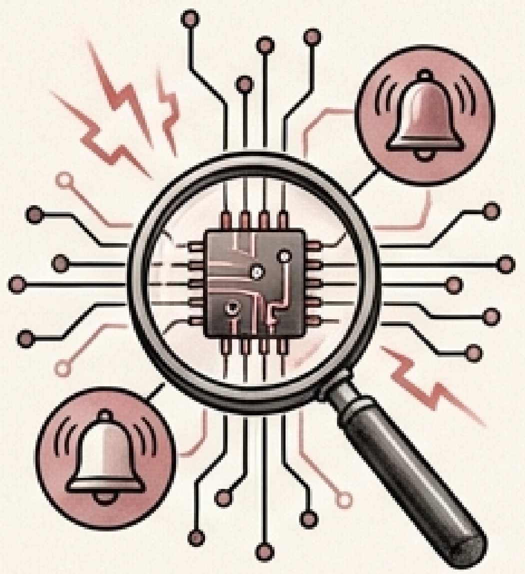


SBOM: A Cornerstone of Supply Chain Transparency



Generate an SBOM on every build:

Create a complete inventory of all components used in your software.

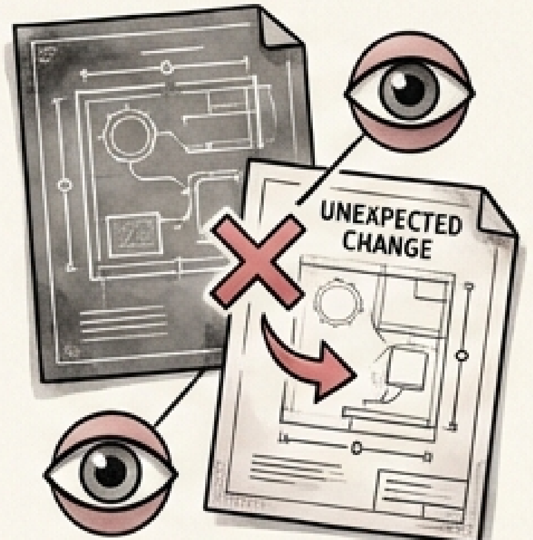


Monitor SBOM for new vulnerabilities continuously:

Track newly disclosed vulnerabilities affecting the components listed in your SBOM.

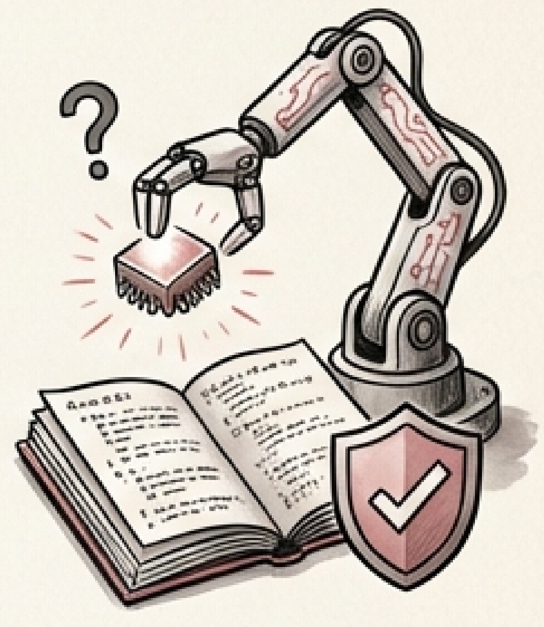
Share SBOM with customers and partners:

Comply with regulatory requirements and enhance trust by providing transparency into your software supply chain. *(Executive Order 14028 requirement)*



SBOM diff between builds detects unexpected dependency changes:

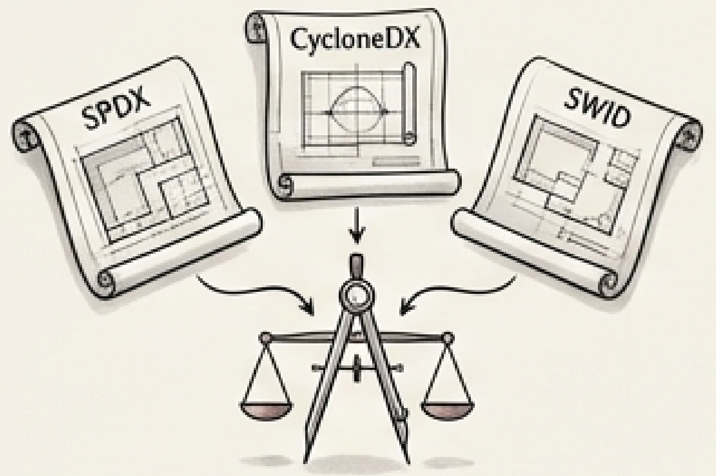
Quickly identify supply chain tampering by comparing SBOMs across builds.



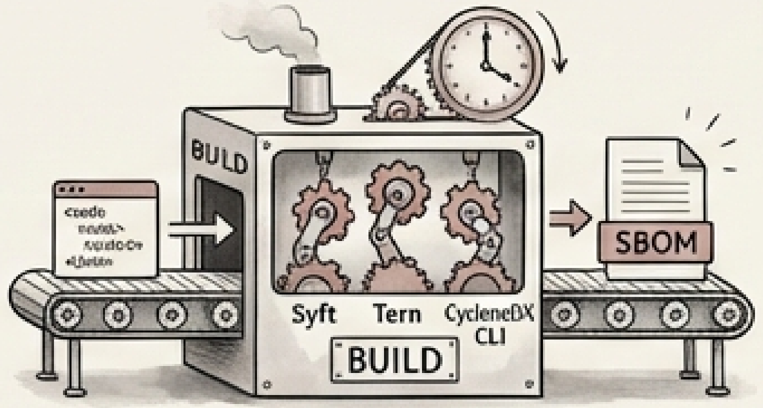
AI-generated code may silently introduce dependencies not in your approved list:

SBOM catches drift and unauthorized component usage.

SBOM Tools and Standards: What You Need to Know

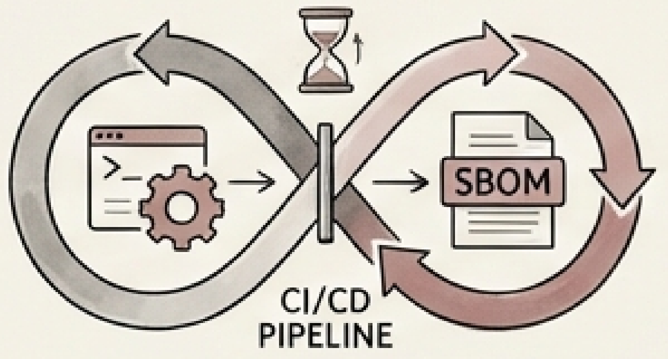


- SPDX, CycloneDX, and SWID are common SBOM formats: Choose a format that aligns with your organization’s needs and industry standards.



- Tools like Syft, Tern, and CycloneDX CLI can automatically generate SBOMs from your builds.

- Consider using a central SBOM repository to manage and share SBOMs across your organization.



- Automate SBOM generation as part of your CI/CD pipeline: Ensure that an SBOM is created for every build.

- When possible, request SBOMs from your vendors to better understand the components included in the software you are using.

Private Registries and Proxies: Gating Your Supply Chain

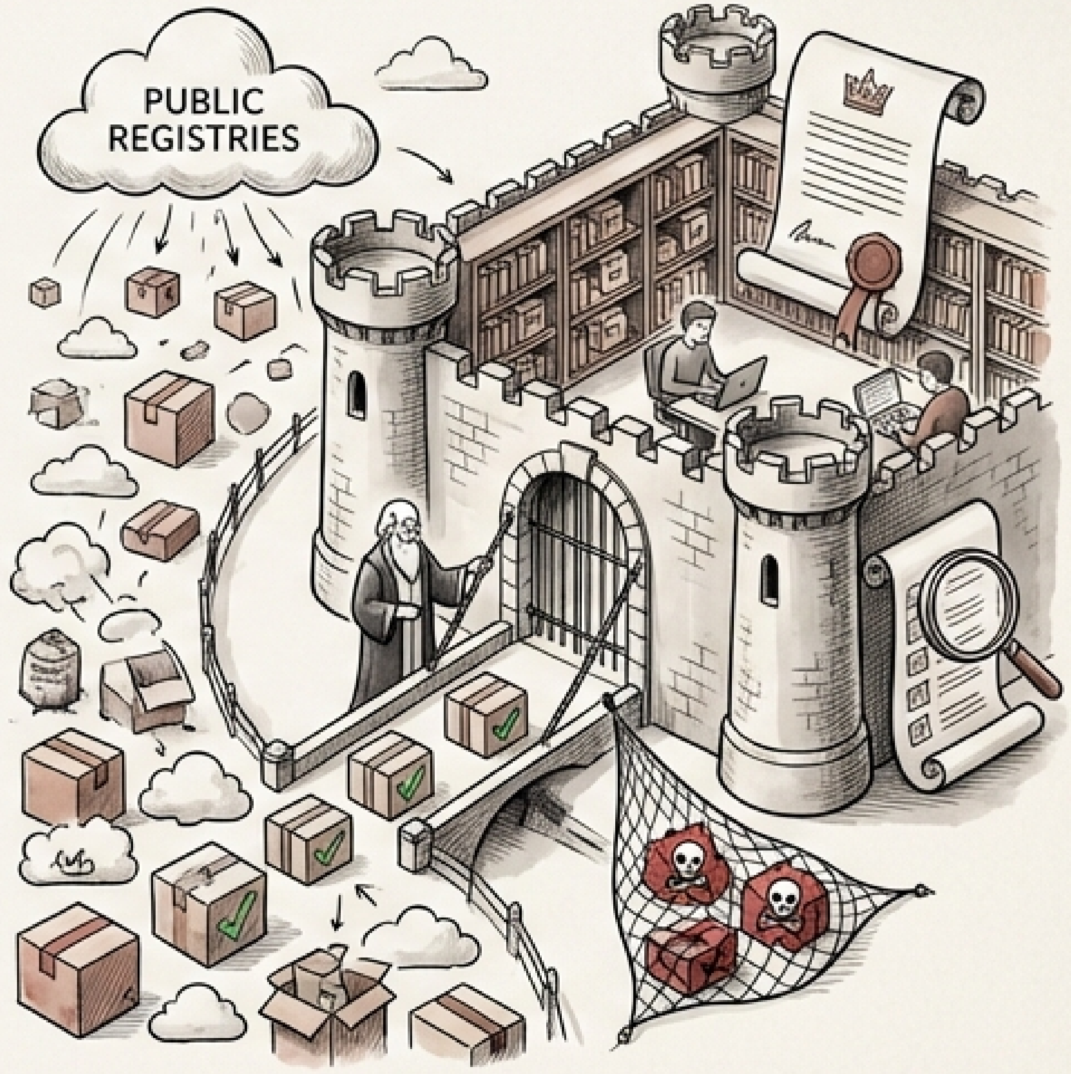
Run a private registry that **proxies** public registries (e.g., Artifactory, Nexus, GitHub Packages) for enhanced control.

Cache approved packages locally: Reduce reliance on public registries and ensure availability of dependencies.

Block known-malicious packages: Prevent developers from downloading and using packages with known vulnerabilities.

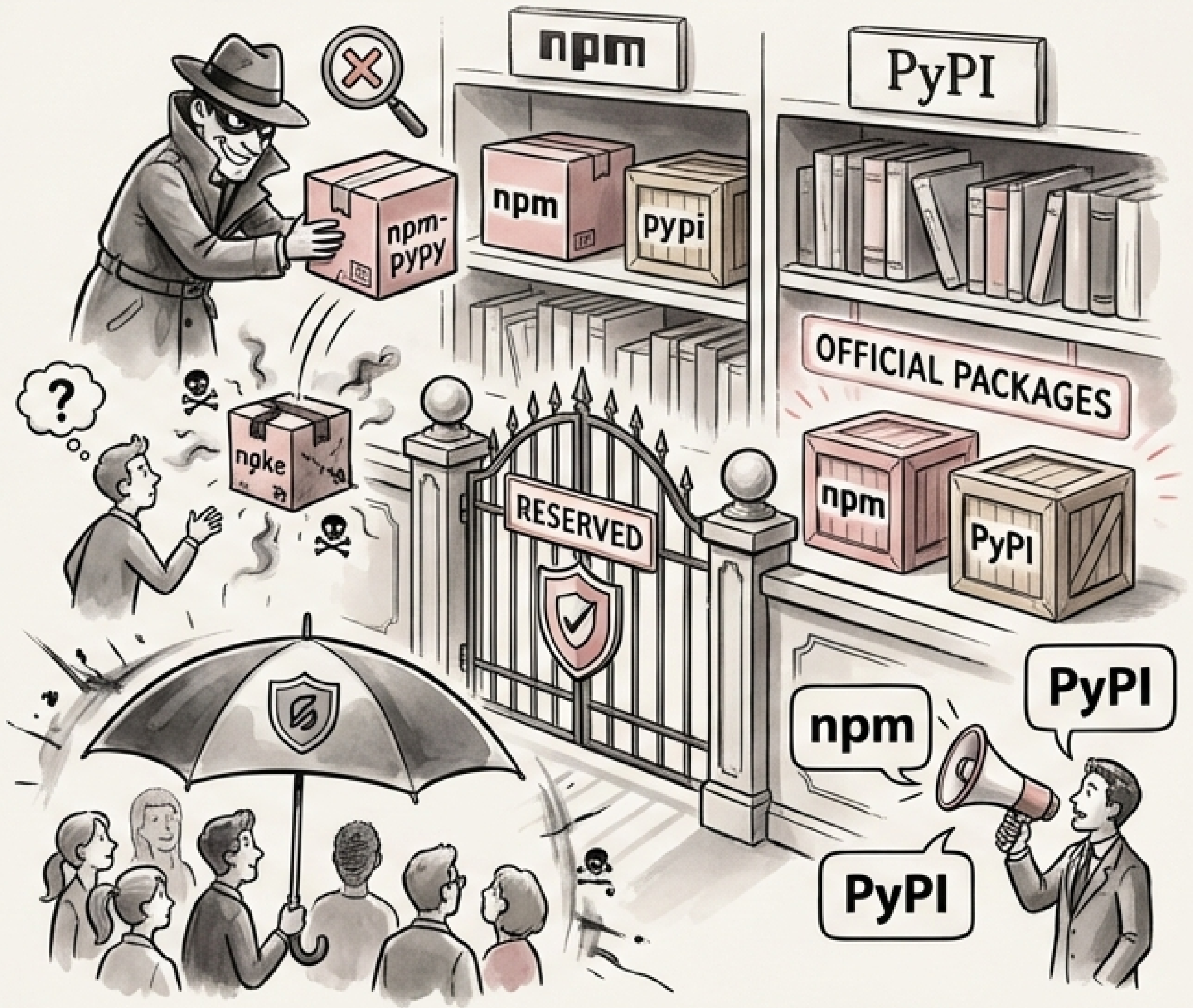
Enforce license policies: Control the use of dependencies based on their licenses to comply with legal requirements.

Audit all package downloads: Track which packages are being used and by whom for improved visibility and accountability.



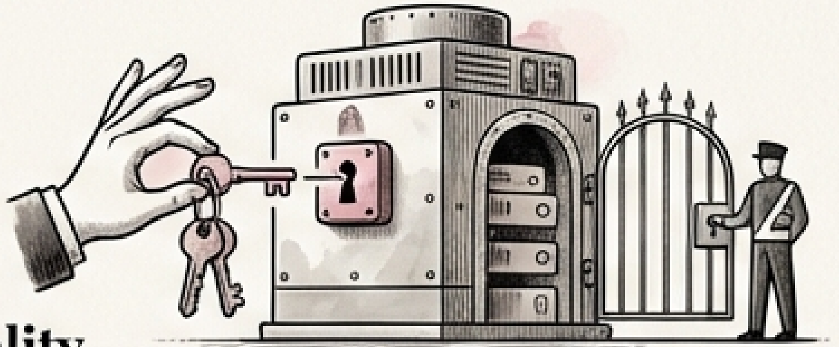
NAMESPACE RESERVATION: PROTECTING YOUR BRAND AND USERS

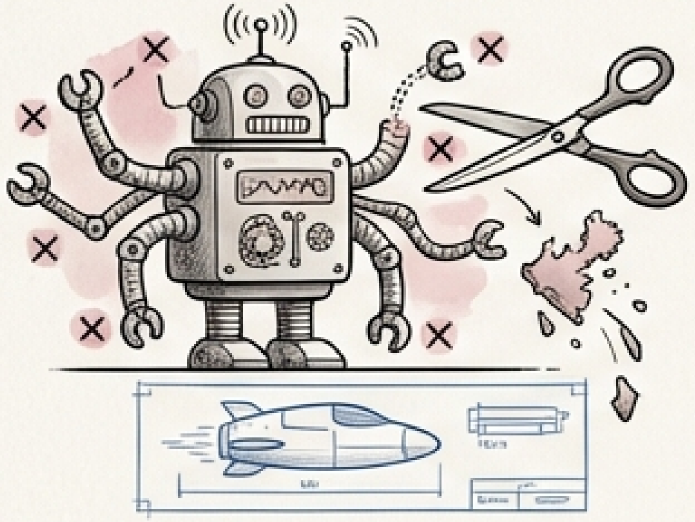
- Register your organization's name on public registries (e.g., npm, PyPI) to prevent typosquatting attacks.
- Typosquatting: Attackers register packages with names similar to legitimate packages to trick developers into installing malicious code.
- Namespace reservation makes it more difficult for attackers to impersonate your organization's packages.
- This is a proactive measure to protect your brand and users from malicious packages.
- Communicate your official package names to developers to prevent confusion.



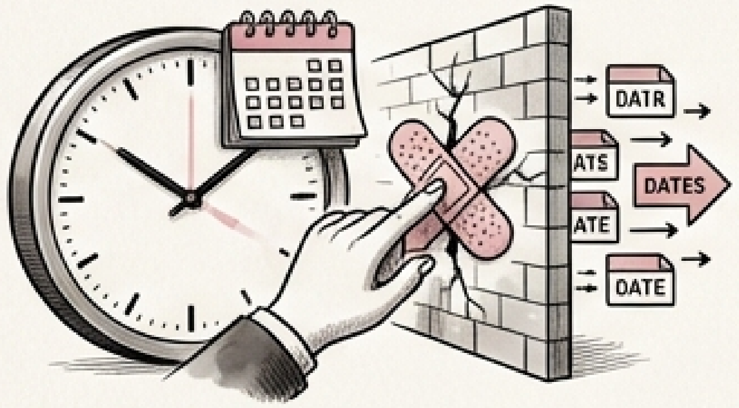
Secure Configuration and Hardening: Reducing Attack Vectors

- **Apply the principle of least privilege to all dependencies:** Grant only the necessary permissions and access rights.

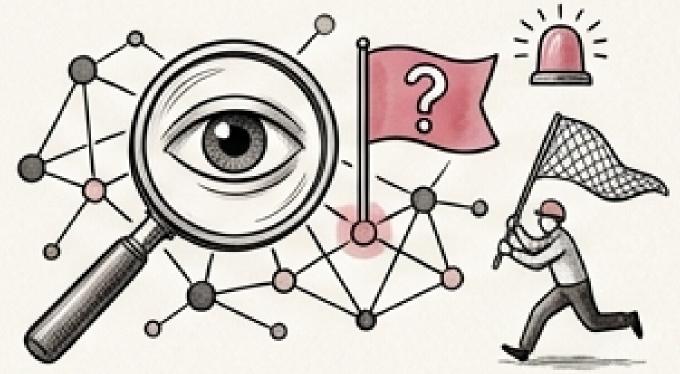




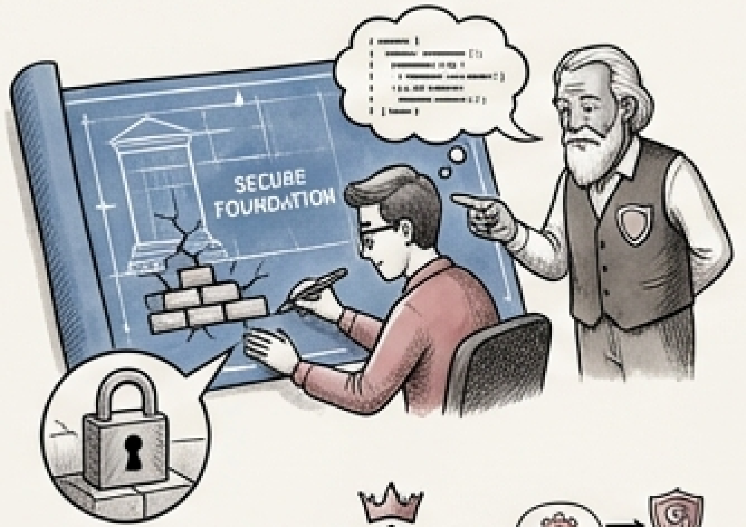
- **Disable unnecessary features and functionality in dependencies:** Reduce the attack surface by removing unused code.
- **Regularly update dependencies to patch security vulnerabilities:** Keep your dependencies up to date to mitigate known risks.



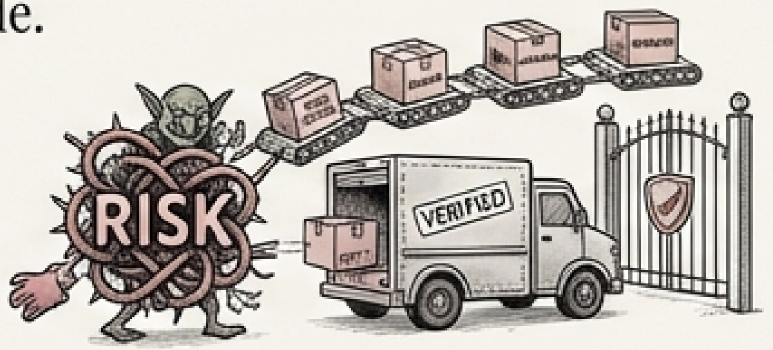
- **Implement strong authentication and authorization mechanisms** for accessing dependencies.
- **Monitor dependencies for suspicious activity:** Detect and respond to potential attacks in a timely manner.



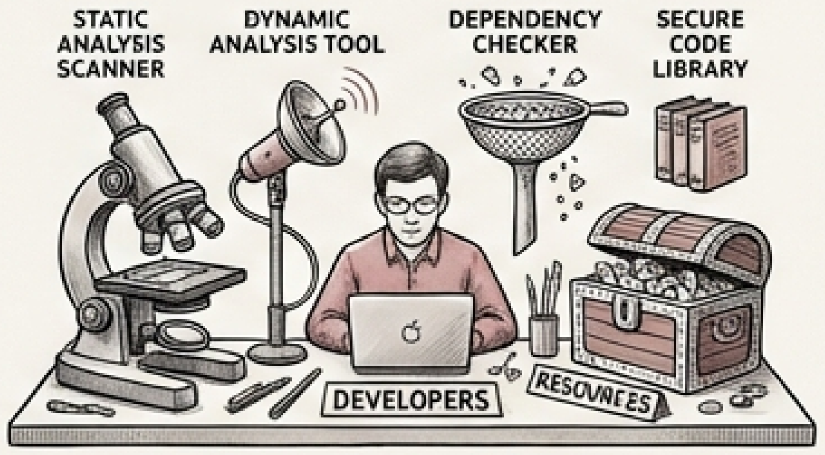
DEVELOPER EDUCATION: BUILDING A SECURITY-AWARE CULTURE



- **Train developers on secure coding practices:** Teach them how to avoid common vulnerabilities and write secure code.

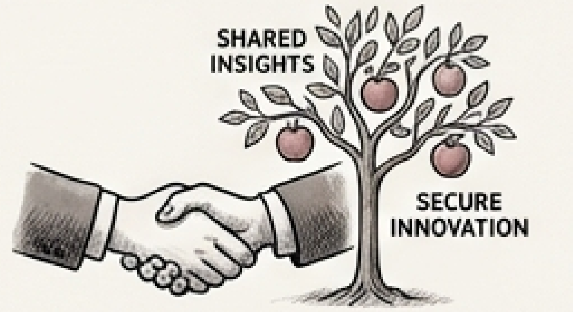


- **Educate developers about supply chain risks:** Make them aware of the potential threats and how to mitigate them.



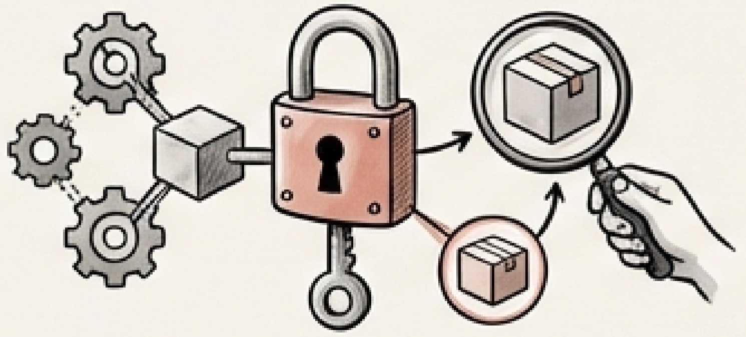
- **Promote a security-first culture:** Encourage developers to prioritize security in all aspects of their work.
- **Provide developers with the tools and resources** they need to build secure software.

- **Foster collaboration between security and development teams:** Encourage open communication and knowledge sharing.

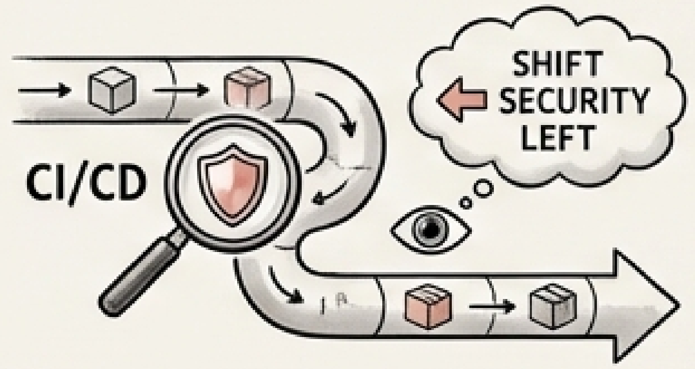


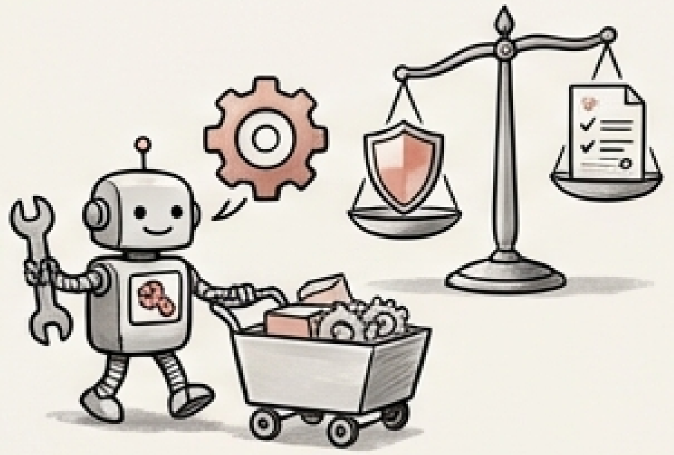
KEY TAKEAWAYS

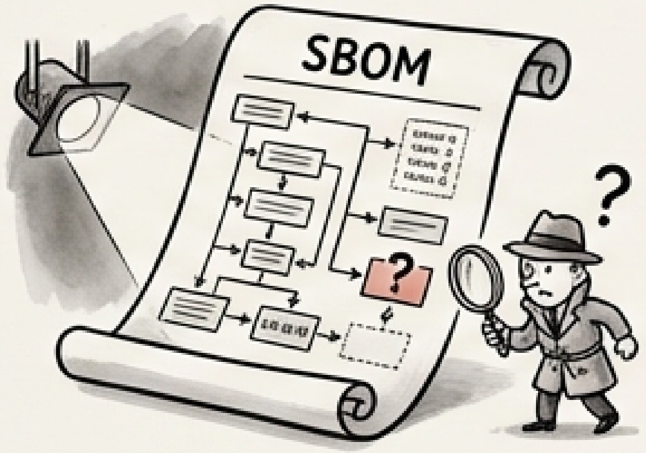
Key Takeaways: Securing Your AI-Augmented Supply Chain



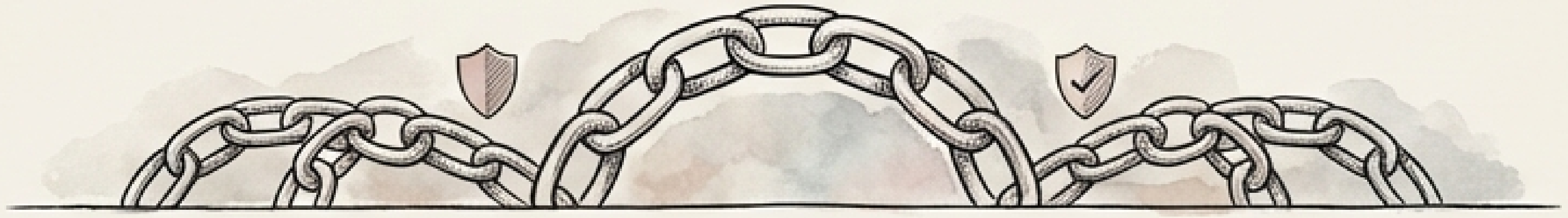
- **Implement robust dependency management practices:** Use lockfiles, review dependencies, and minimize dependencies.
- **Automate dependency updates with tools like Dependabot/Renovate:** Balance security and stability with a well-defined update strategy.
- **Integrate SCA tools into your CI/CD pipeline:** Shift security left and continuously monitor for vulnerabilities.
- **Generate and manage SBOMs:** Gain visibility into your software supply chain and detect unexpected changes.
- **Use private registries and proxies:** Control access to dependencies and enforce security policies.







Next Steps: Building a More Secure Future



- **Conduct a security assessment of your software supply chain:** Identify vulnerabilities and areas for improvement.
- **Implement the recommendations outlined in this presentation:** Take concrete steps to secure your supply chain.
- **Stay informed about the latest supply chain threats and vulnerabilities:** Continuously monitor the threat landscape.
- **Share this information with your colleagues and teams:** Promote a security-aware culture throughout your organization.
- **Invest in security tools and training:** Provide your teams with the resources they need to build secure software.

Thank You

- Questions?