AI-Augmented Development: The Security Imperative

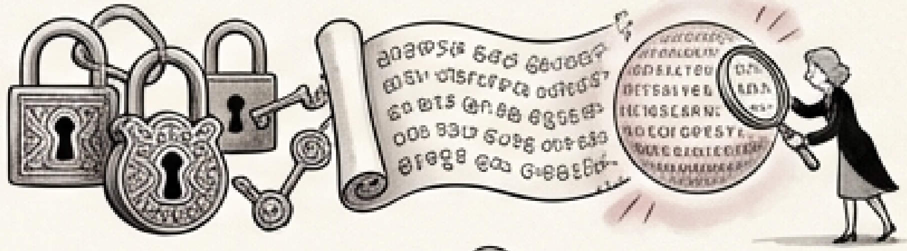
AI-Augmented Development: The Security Imperative

- AI tools in development offer powerful acceleration but can introduce significant security vulnerabilities.
- A core defense: prioritize vetted, actively maintained security libraries over custom implementations.
- AI frequently suggests unvetted, deprecated, or even completely non-existent libraries, increasing risk.
- Focusing on trusted foundations is crucial for mitigating the novel security challenges posed by AI-assisted coding.
- Using approved security libraries is a high-impact secure coding practice.

Approved Security Libraries by Domain: A Developer's Toolkit

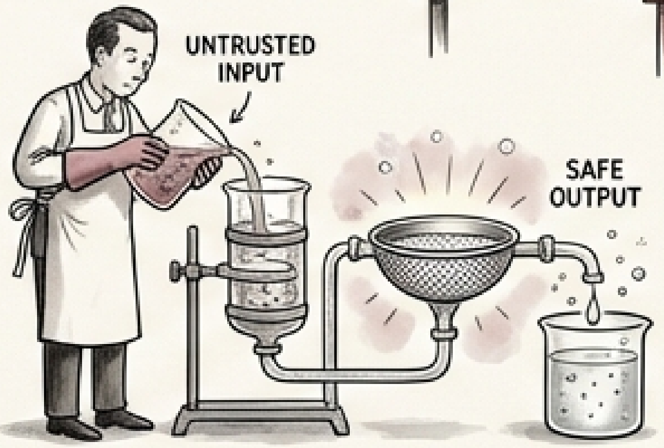
- **Authentication:** Passport.js (Node), Spring Security (Java), Django auth (Python), ASP.NET Identity (.NET), NextAuth.js (React).

- **Cryptography:** libsodium/NaCl, OpenSSL/BoringSSL, Bouncy Castle (Java), Web Crypto API (browser), PyNaCl (Python).



- **Input Validation:** Joi/Zod (JS/TS), Cerberus/Pydantic (Python), Bean Validation (Java), FluentValidation (.NET).

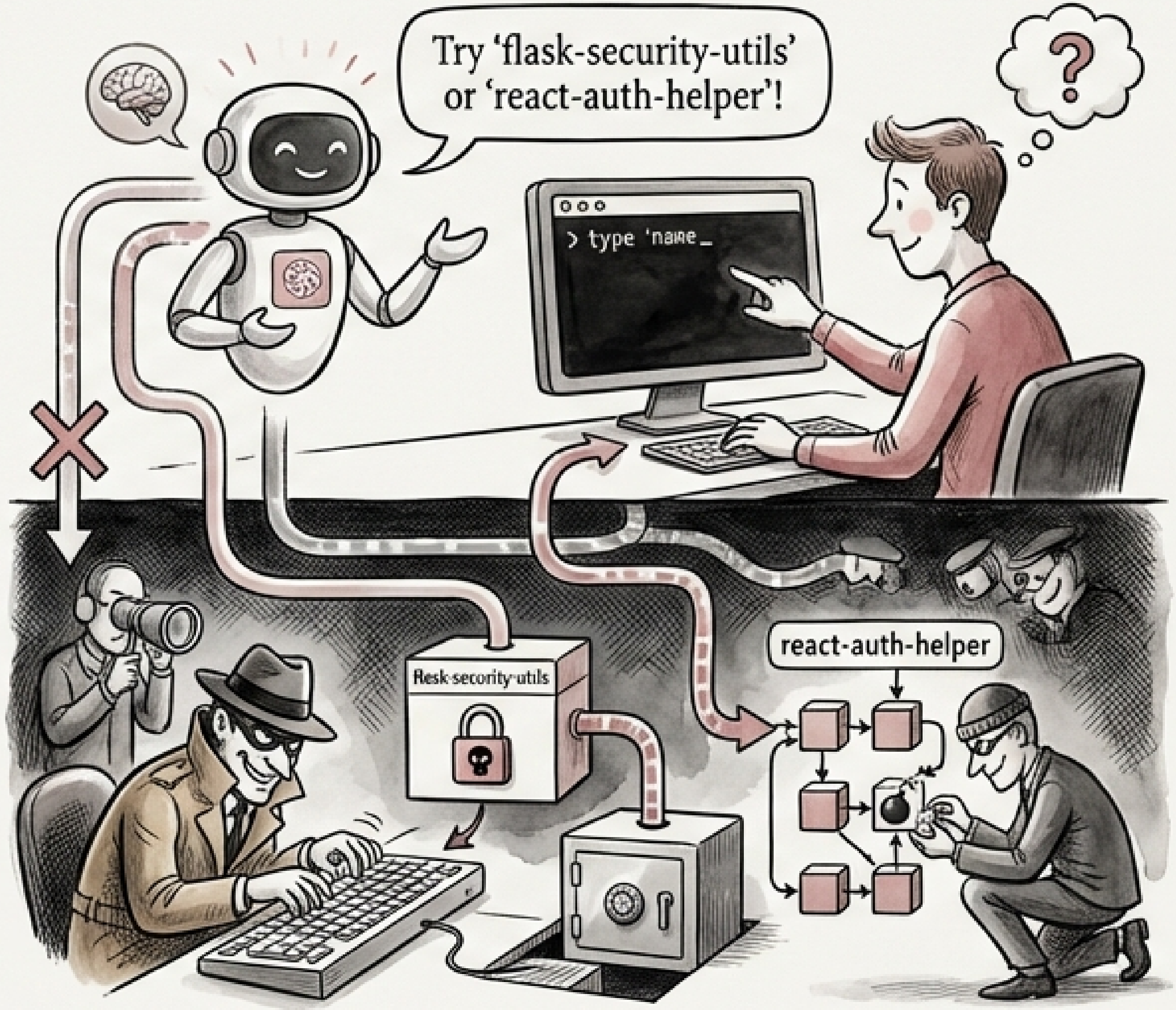
- **Output Encoding:** DOMPurify (JS), OWASP Java Encoder, bleach (Python).

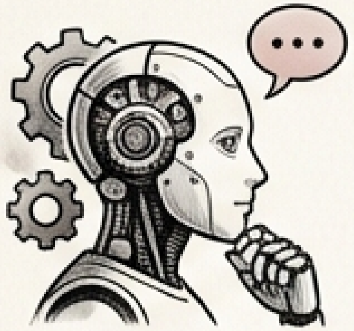


- **CSRF:** csurf (Express), Django middleware, Spring Security CSRF.

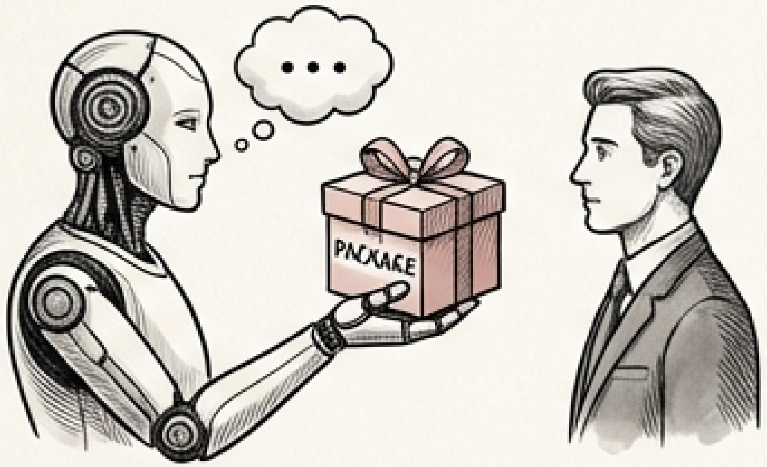
The Slopsquatting Epidemic: A New AI-Driven Attack Vector

- 19.7% of AI-recommended packages do not exist on any official registry.
- Attackers are actively monitoring AI suggestion patterns to identify potential ‘slopsquatting’ opportunities.
- Attackers register phantom package names with malicious payloads, hoping developers will mistakenly install them.
- Example: AI suggests `flask-security-utils` (non-existent); an attacker registers it with a credential-stealing backdoor.
- Example: AI suggests `react-auth-helper` (non-existent); an attacker registers it with a supply chain compromise.



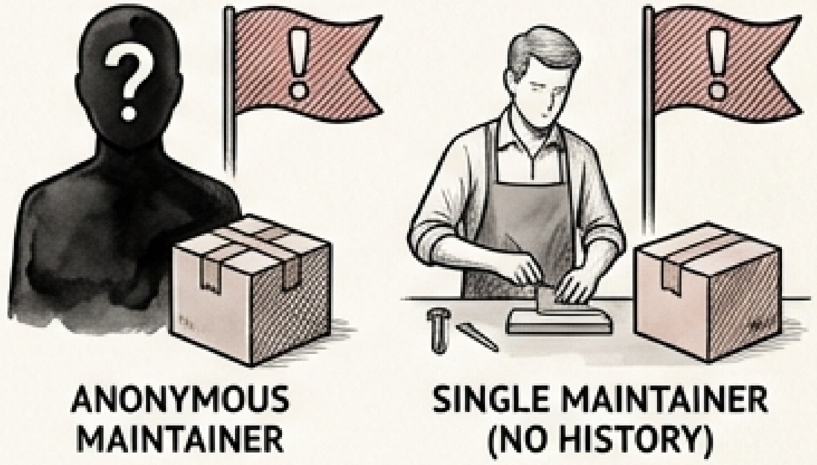


Dependency Verification Workflow: Mitigating AI-Driven Risks



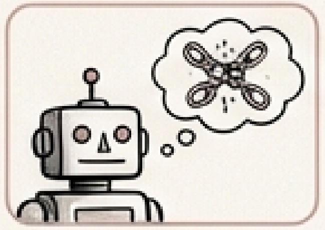
- **Step 1:** AI suggests

- **Step 3:** Check the creation date – packages created in the last 90 days are suspicious.
- **Step 4:** Check maintainer identity – anonymous or single maintainer with no history is a red flag.
- **Step 5:** Check **download count** and **community activity** – legitimate packages have organic adoption curves.



LEGITIMATE
SUSPICIOUS
DOWNLOAD COUNT & COMMUNITY ACTIVITY

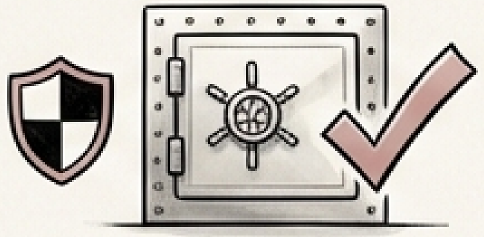
No Exceptions: Custom Crypto is Always Wrong



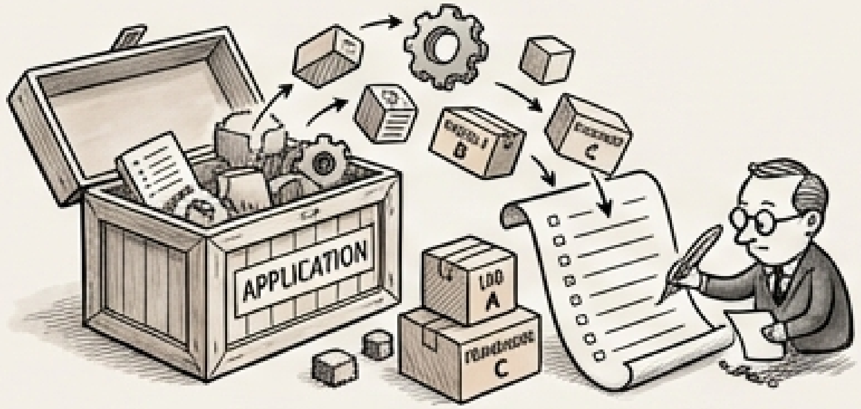
- AI tools regularly generate custom encryption functions, which are almost always flawed and insecure.
- Examples of insecure custom crypto: XOR-based schemes, ECB mode usage, Math.random() for security tokens, base64 masquerading as encryption, homegrown key derivation.

- Every single one of these custom implementations is inherently broken and vulnerable.

- Policy: zero tolerance for custom cryptographic implementations, regardless of their source.
- Use approved, vetted cryptographic libraries exclusively for all encryption needs.

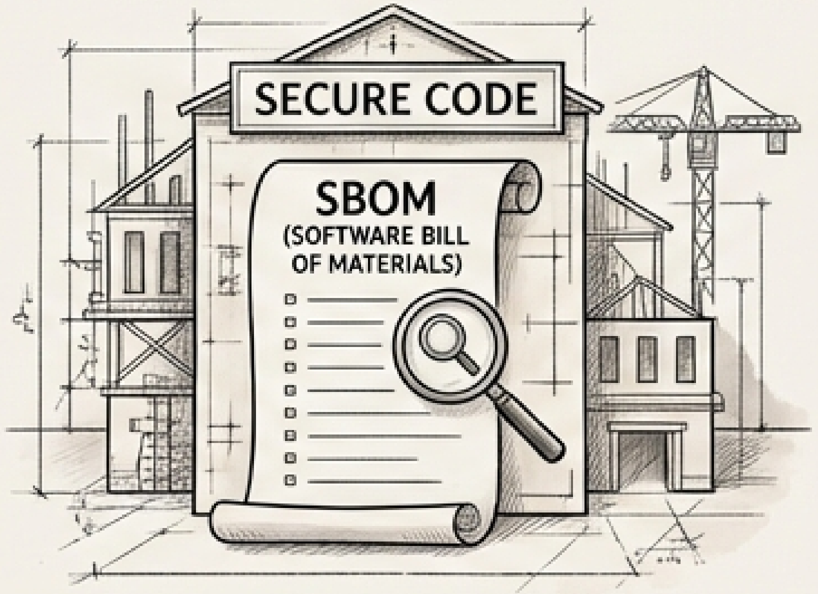


SBOM Generation: Building a Foundation for Secure Code

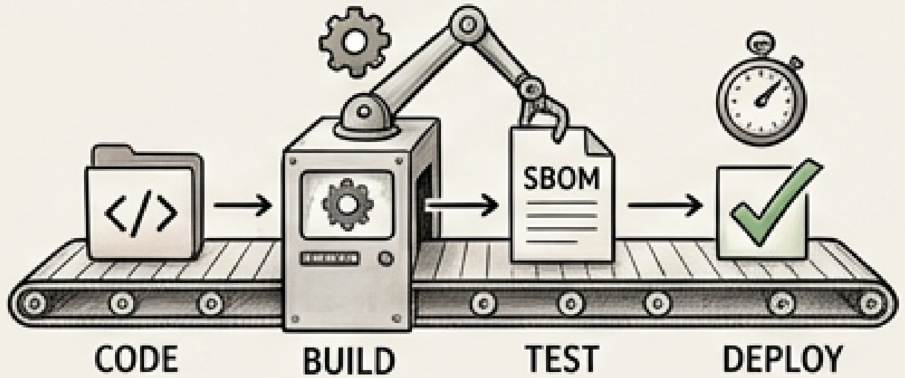


- A Software Bill of Materials (SBOM) is a complete inventory of every component in your application.

- Standard SBOM formats: **SPDX 2.3** and **CycloneDX 1.5.**

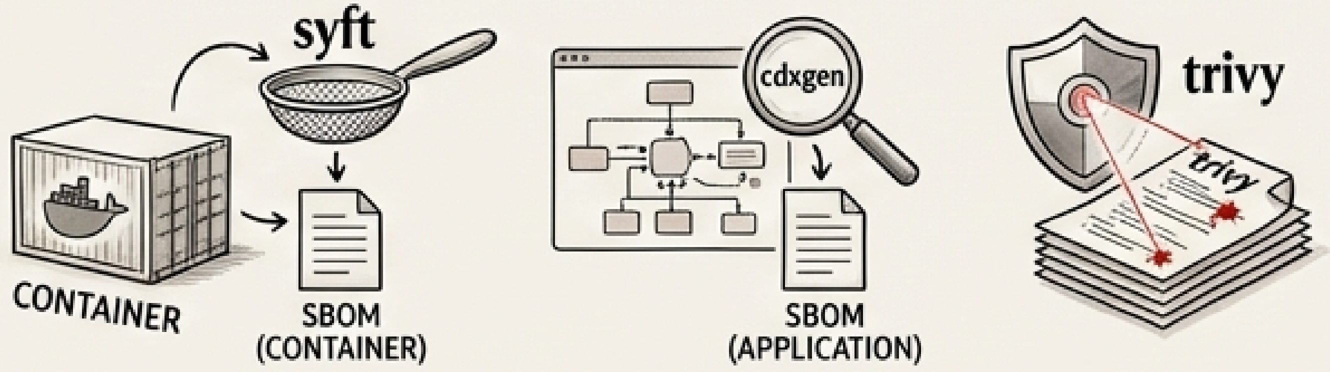


- Generate SBOMs automatically in the CI pipeline on every build to ensure accuracy.

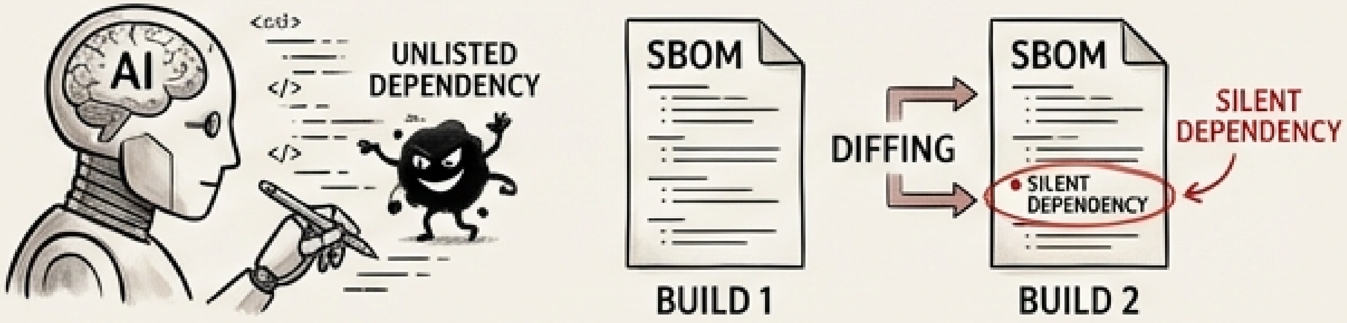


- Standard SBOM formats:

- Tools for SBOM generation: **syft** (container SBOM), **cdxgen** (application SBOM), **trivy** (vulnerability scanning against SBOM)



- AI-generated code may silently introduce unlisted dependencies; SBOM diffing between builds can catch this



Component Lifecycle Management: From Selection to Deprecation

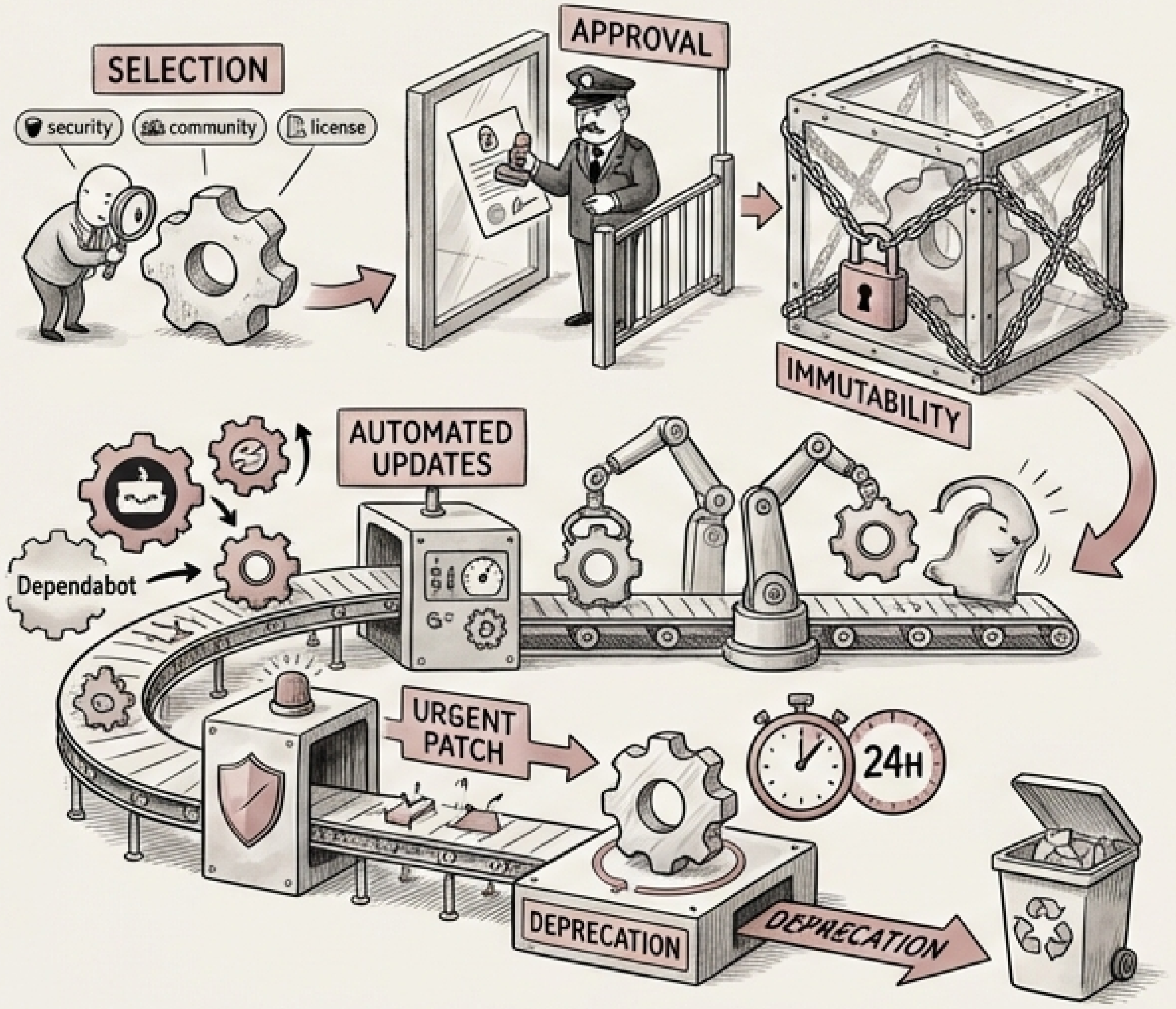
Selection: Use security evaluation criteria, maintainer assessment, community health metrics, and license review to choose components.

Approval: Security team sign-off is required for all security-critical components.

Version pinning: Use exact versions in lockfiles and verify hashes to ensure immutability.

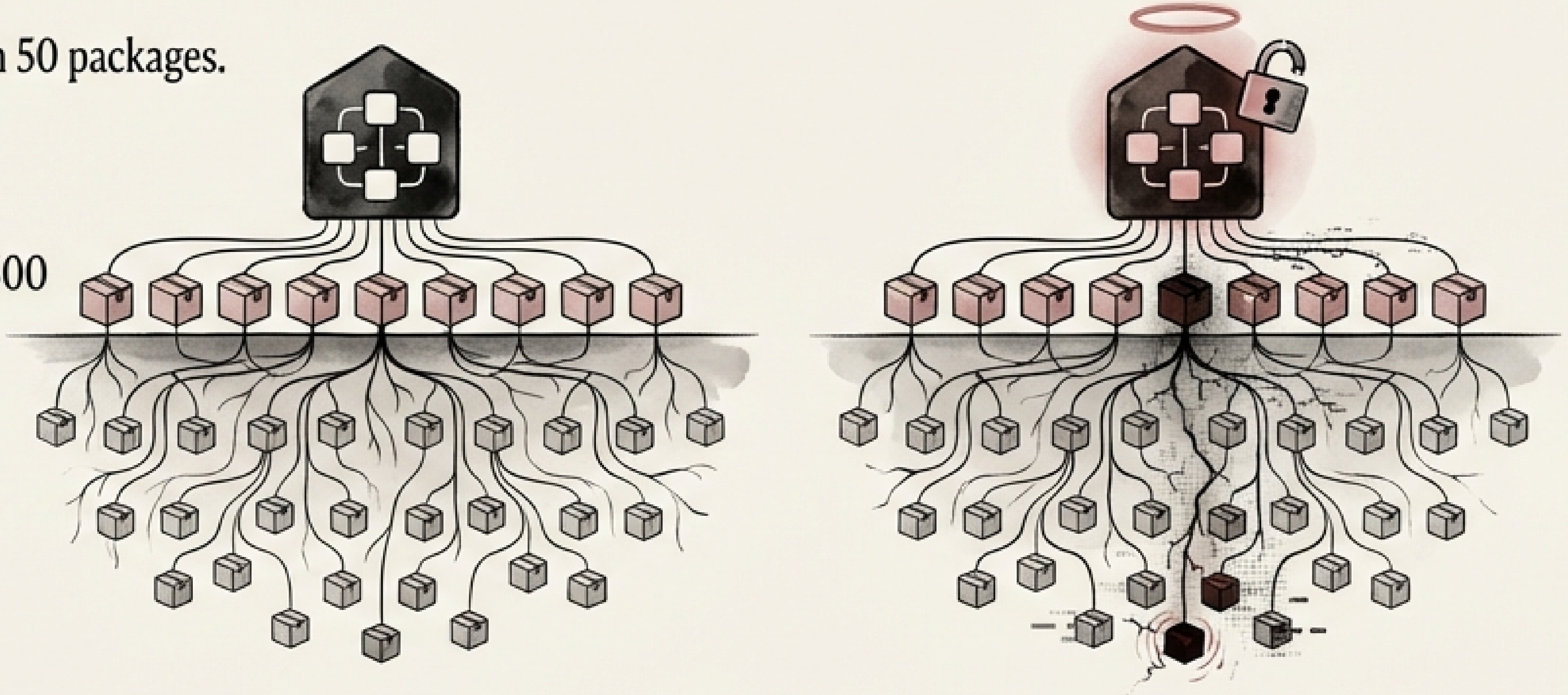
Update cadence: Automate updates with tools like Dependabot/Renovate.

Prioritize security-critical patches, applying them within 24 hours of release.



TRANSITIVE DEPENDENCY RISK: THE HIDDEN DANGERS

- Your application depends directly on 50 packages.
- Those 50 packages may depend on 500 packages (transitive dependencies).
- A single compromised transitive dependency can affect the entire

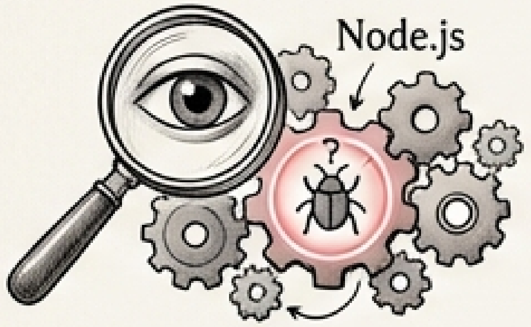


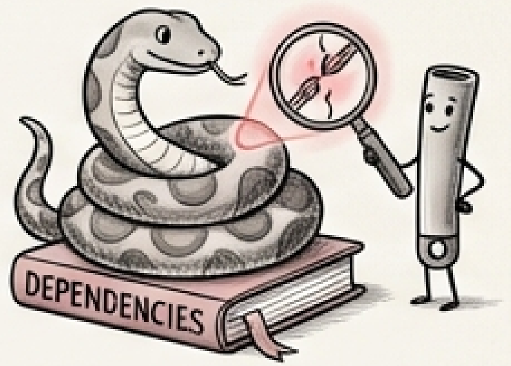
REAL EXAMPLES

- Real example: `event-stream` (2018) – malicious code injected into a transitive dependency, compromised bitcoin wallets.

- Real example: `ua-parser-js` (2021) – crypto-mining malware injected into a popular library.

Dependency Scanning Tools: Identifying Vulnerabilities Early





- **npm audit:** Identifies vulnerabilities in Node.js dependencies.

- **pip-audit:** Identifies vulnerabilities in Python dependencies.

- **Trivy:** Comprehensive vulnerability scanner for containers, file systems, and cloud infrastructure. Can scan SBOMs.

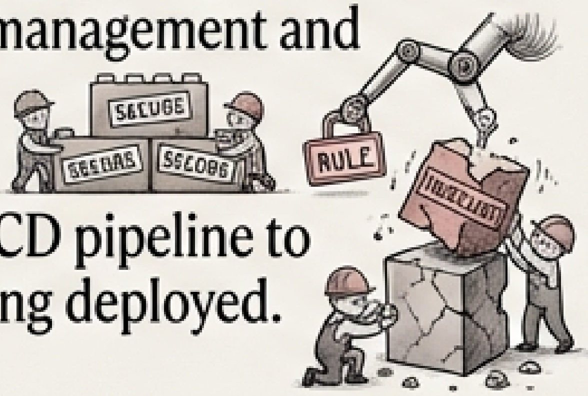
- **Snyk:** Developer security platform that identifies vulnerabilities in code, dependencies, and containers.

- **OWASP Dependency-Check:** Open-source tool for identifying known vulnerabilities in project dependencies.

Semgrep: Automating Security Rule Enforcement

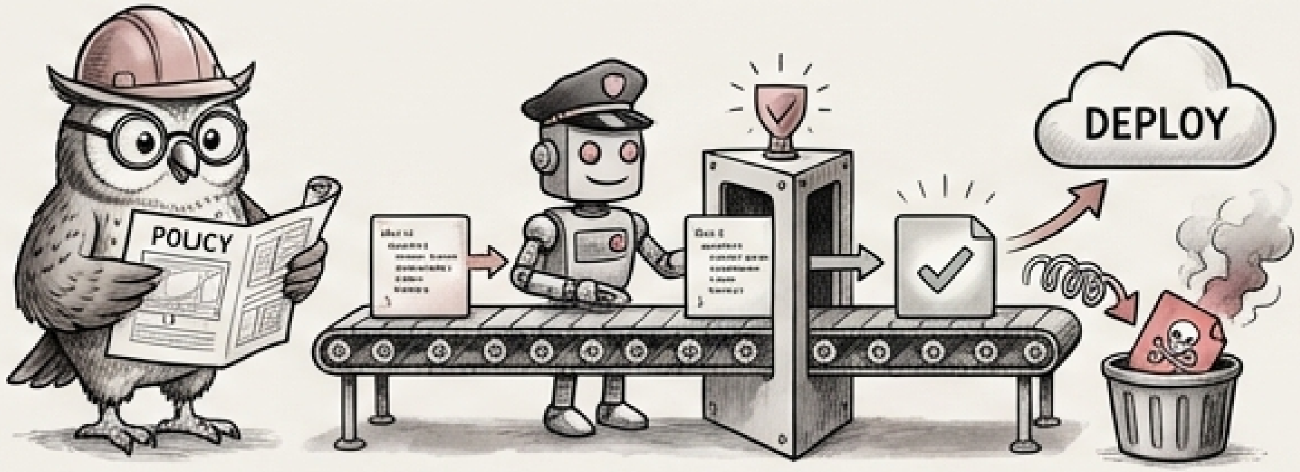
- Semgrep is a fast, open-source static analysis tool for finding bugs and enforcing code standards.

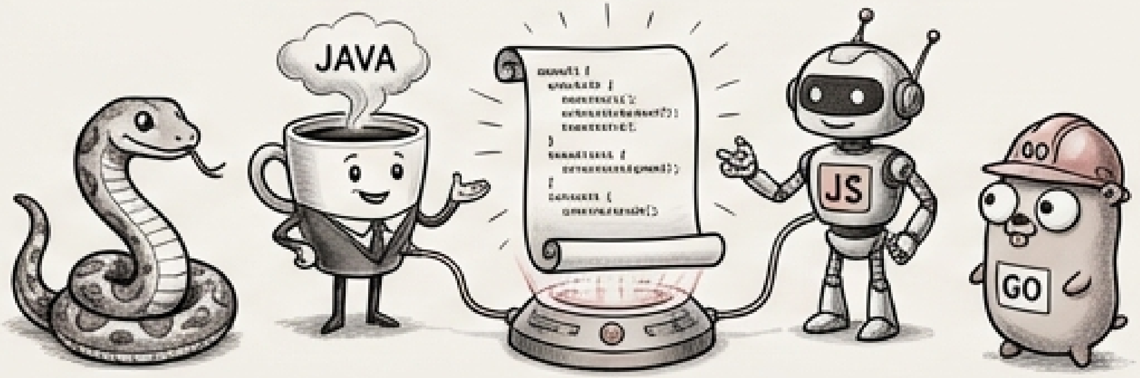
- Use Semgrep to automatically detect custom cryptographic implementations and other prohibited patterns.
- Create custom Semgrep rules to enforce specific security policies related to dependency management and secure coding practices.



- Integrate Semgrep into the CI/CD pipeline to prevent insecure code from being deployed.
- Semgrep can analyze code in various languages, including Python, Java, JavaScript, and Go.

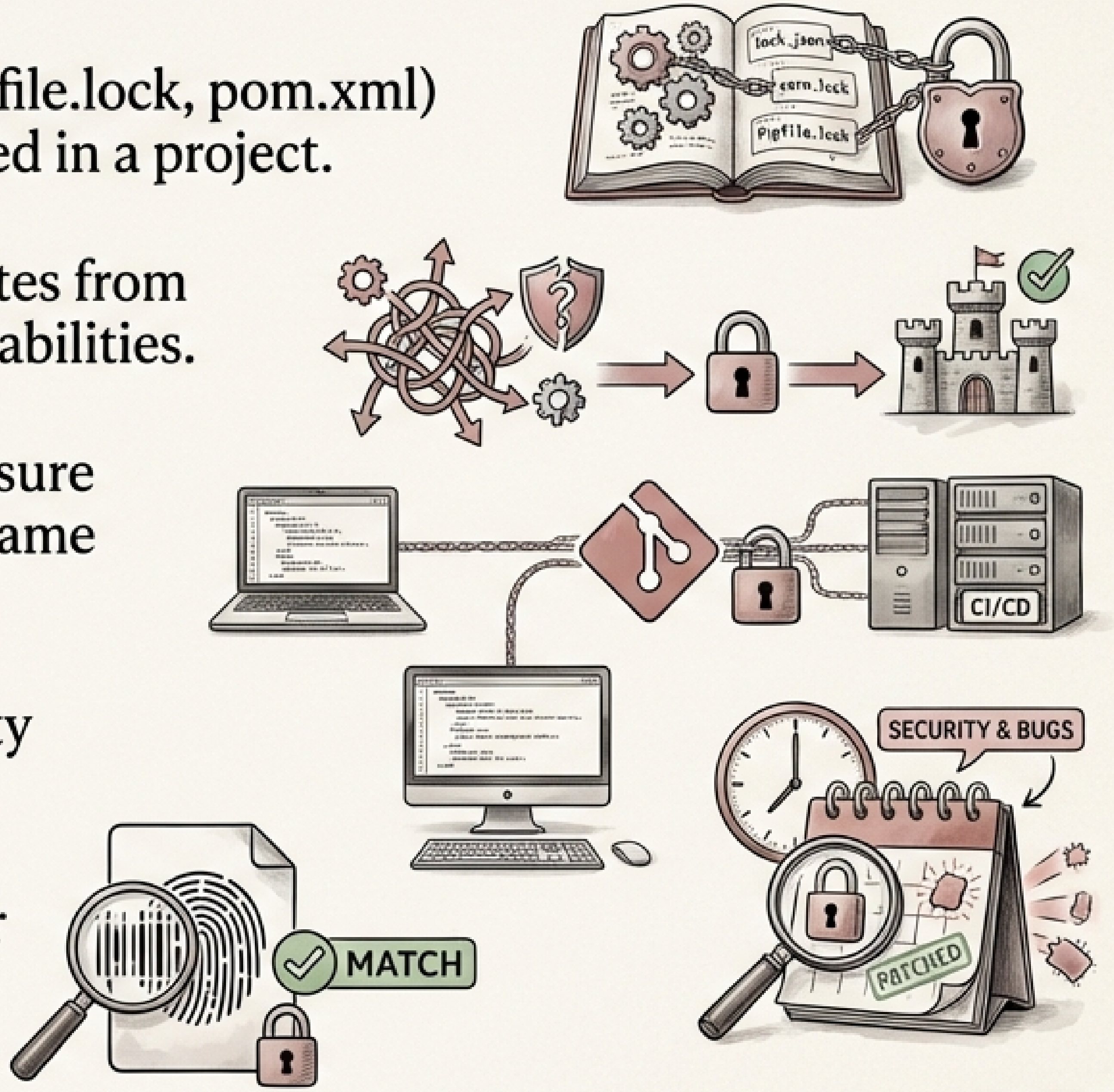
- Use Semgrep to automatically detect custom cryptographic implementations other prohibited patterns.





Lockfiles: Ensuring Reproducible Builds

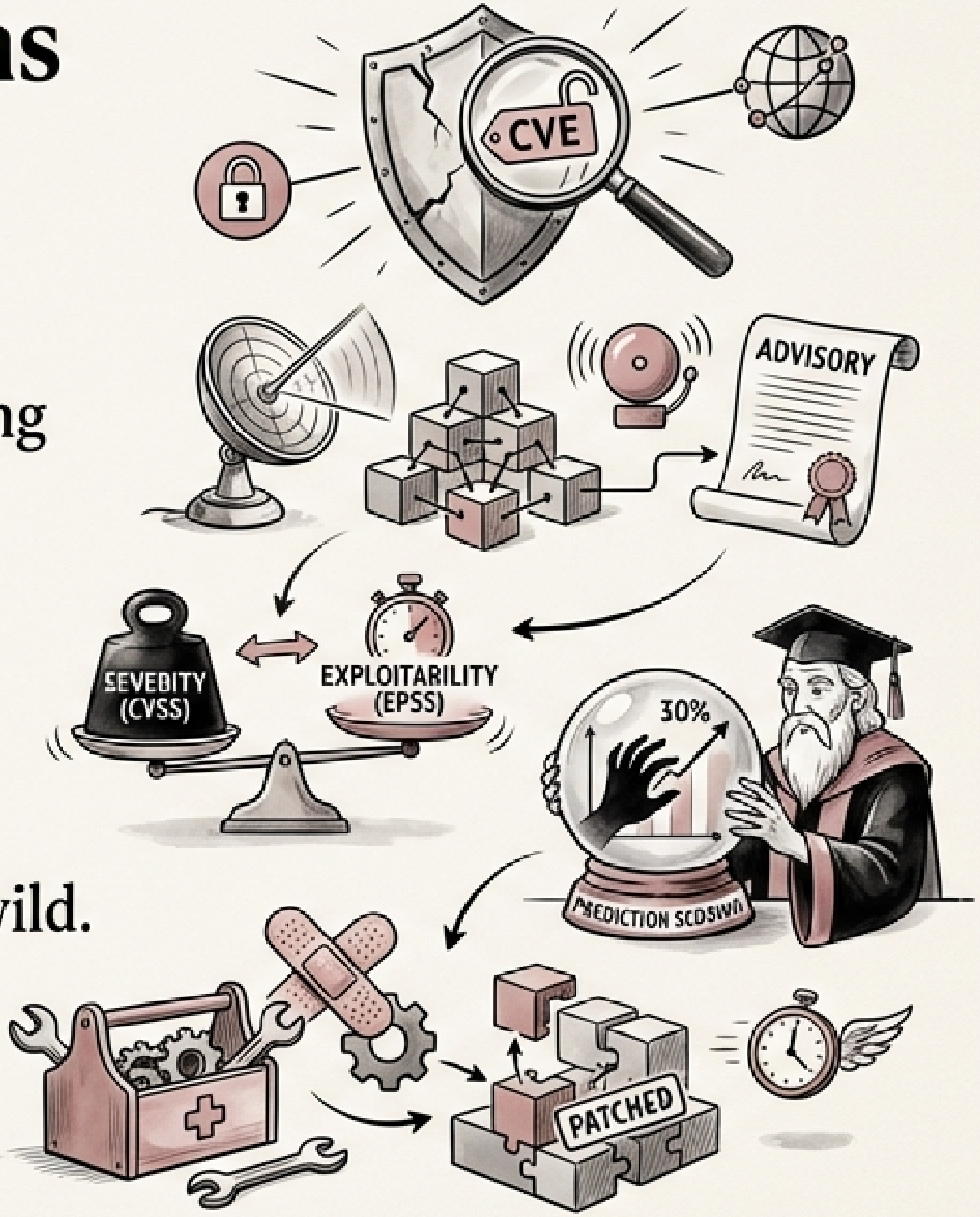
- Lockfiles (e.g., package-lock.json, yarn.lock, Pipfile.lock, pom.xml) record the exact versions of all dependencies used in a project.
- Lockfiles prevent unexpected dependency updates from introducing breaking changes or security vulnerabilities.
- Always commit lockfiles to version control to ensure that all developers and CI/CD systems use the same dependency versions.
- Regularly update lockfiles to incorporate security patches and bug fixes.
- Verify the integrity of lockfiles by checking their hashes to prevent tampering.



NEW YORKER
EDITORIAL ILLUSTRATION

Responding to CVEs: Prioritization and Remediation Uditeraving rɒmutheroup systems

- CVEs (Common Vulnerabilities and Exposures) are publicly disclosed security vulnerabilities.
- Continuously monitor dependencies for new CVEs using vulnerability scanning tools and security advisories.
- Prioritize CVEs based on severity (CVSS score) and exploitability (EPSS score).
- EPSS (Exploit Prediction Scoring System) predicts the likelihood that a vulnerability will be exploited in the wild.
- Apply security patches and update dependencies to address CVEs promptly.



License Compliance: Avoiding Legal Pitfalls

- Different software licenses impose different obligations on users, such as attribution requirements, restrictions on commercial use, or requirements to share source code.

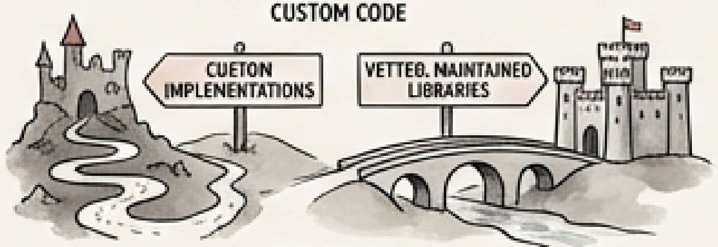
- Common open-source licenses: MIT, Apache 2.0, GPL, BSD.

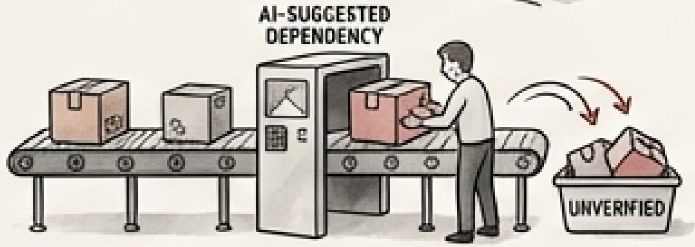
- Failing to comply with license terms can lead to legal consequences.

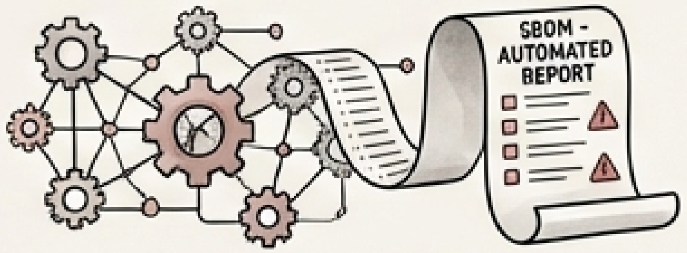
- Use **license scanning tools** to identify the licenses of all dependencies in a project.

- Review license terms carefully and ensure that they are compatible with the project’s goals and business requirements.

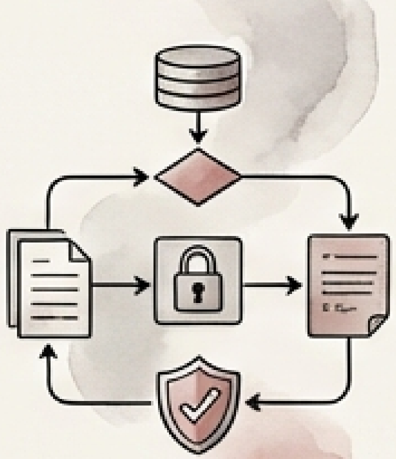
AI AND SECURITY LIBRARIES: KEY TAKEAWAYS



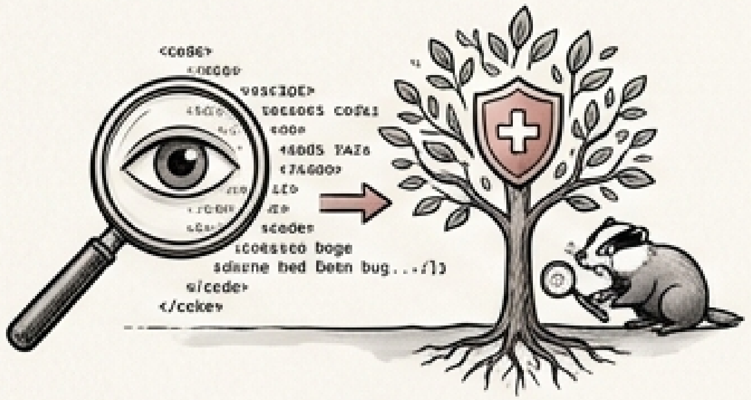




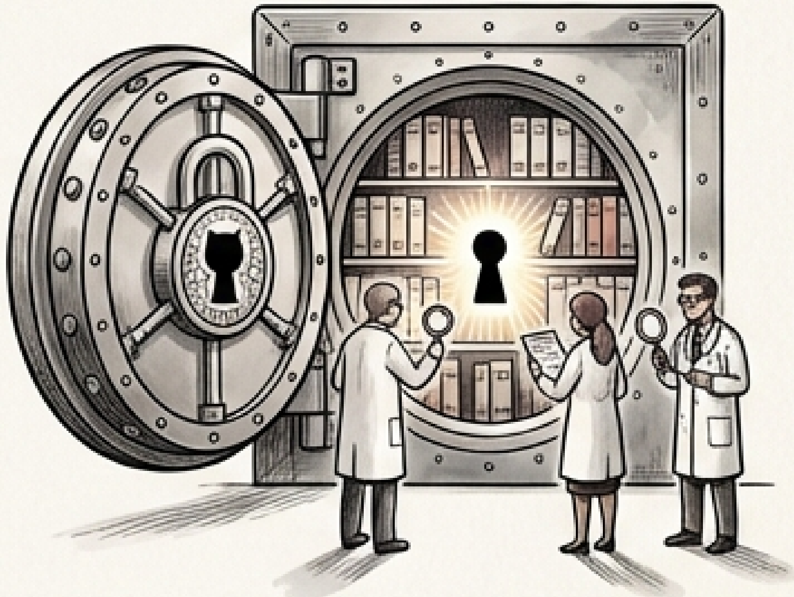
- AI-augmented development introduces new security risks that must be addressed proactively.
- Prioritize vetted, maintained security libraries over custom implementations to reduce vulnerabilities.
- Implement a mandatory dependency verification workflow for all AI-suggested packages.
- Enforce a strict policy against custom cryptographic implementations.
- Generate SBOMs automatically to track application dependencies and identify potential vulnerabilities.



Resources and Further Learning



- OWASP (Open Web Application Security Project): https://owasp.org/
- NIST (National Institute of Standards and Technology) **Cybersecurity Framework:** https://www.nist.gov/cyberframework
- **Snyk:** https://snyk.io/
- **GitHub Security Lab:** https://securitylab.github.com/
- Your internal security team and documentation



Thank You

Questions?