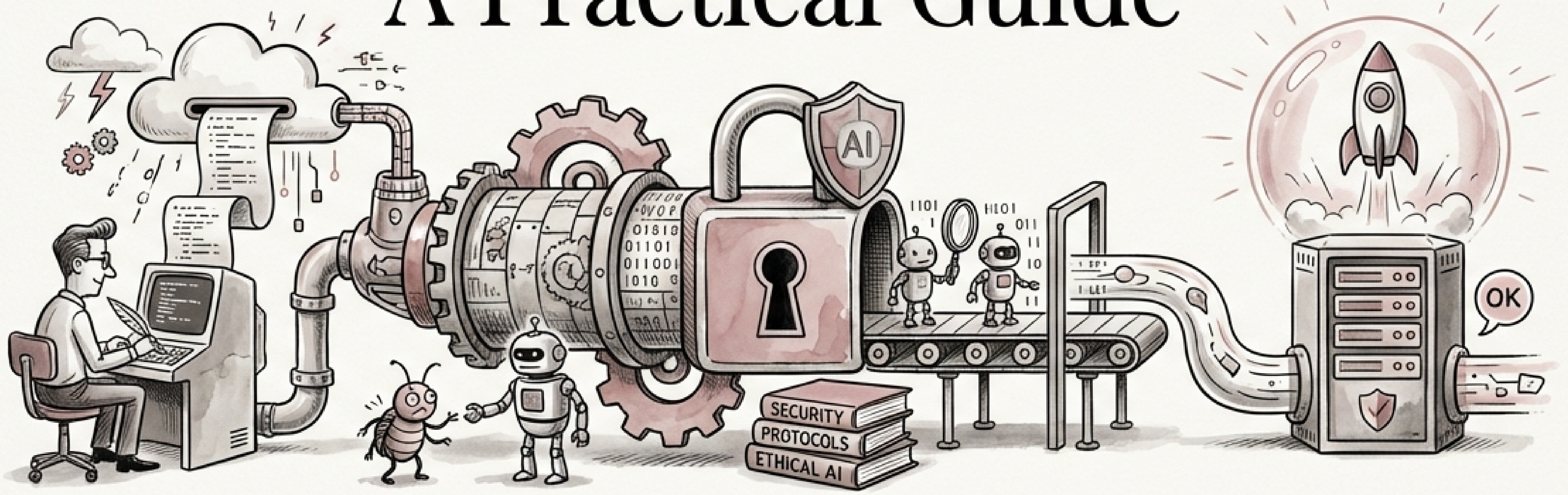
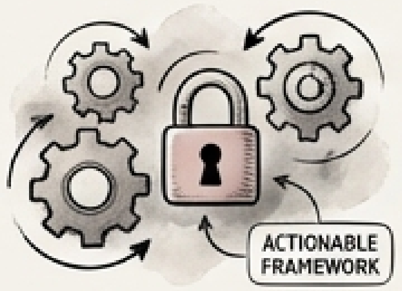
Securing the AI-Augmented Development Lifecycle: A Practical Guide



Securing the AI-Augmented Development Lifecycle: A Practical Guide

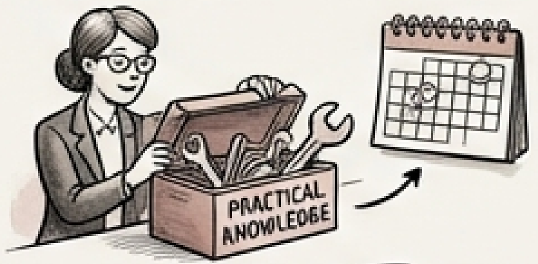


1. This module provides an actionable framework for securing your software development lifecycle.

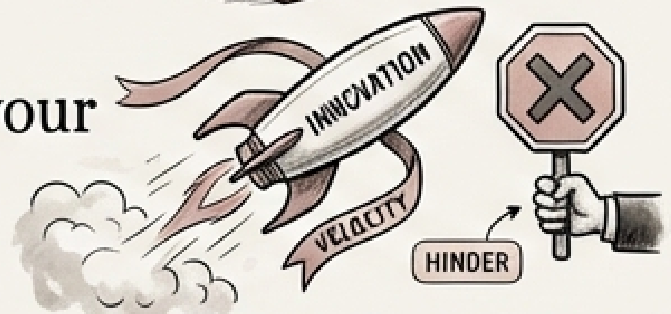
2. We'll be using CIS Controls v8 Control Group 16 as our organizing structure.

3. The core focus is on adapting these controls to modern AI-augmented development teams.

4. You'll gain practical knowledge applicable to your daily workflow.

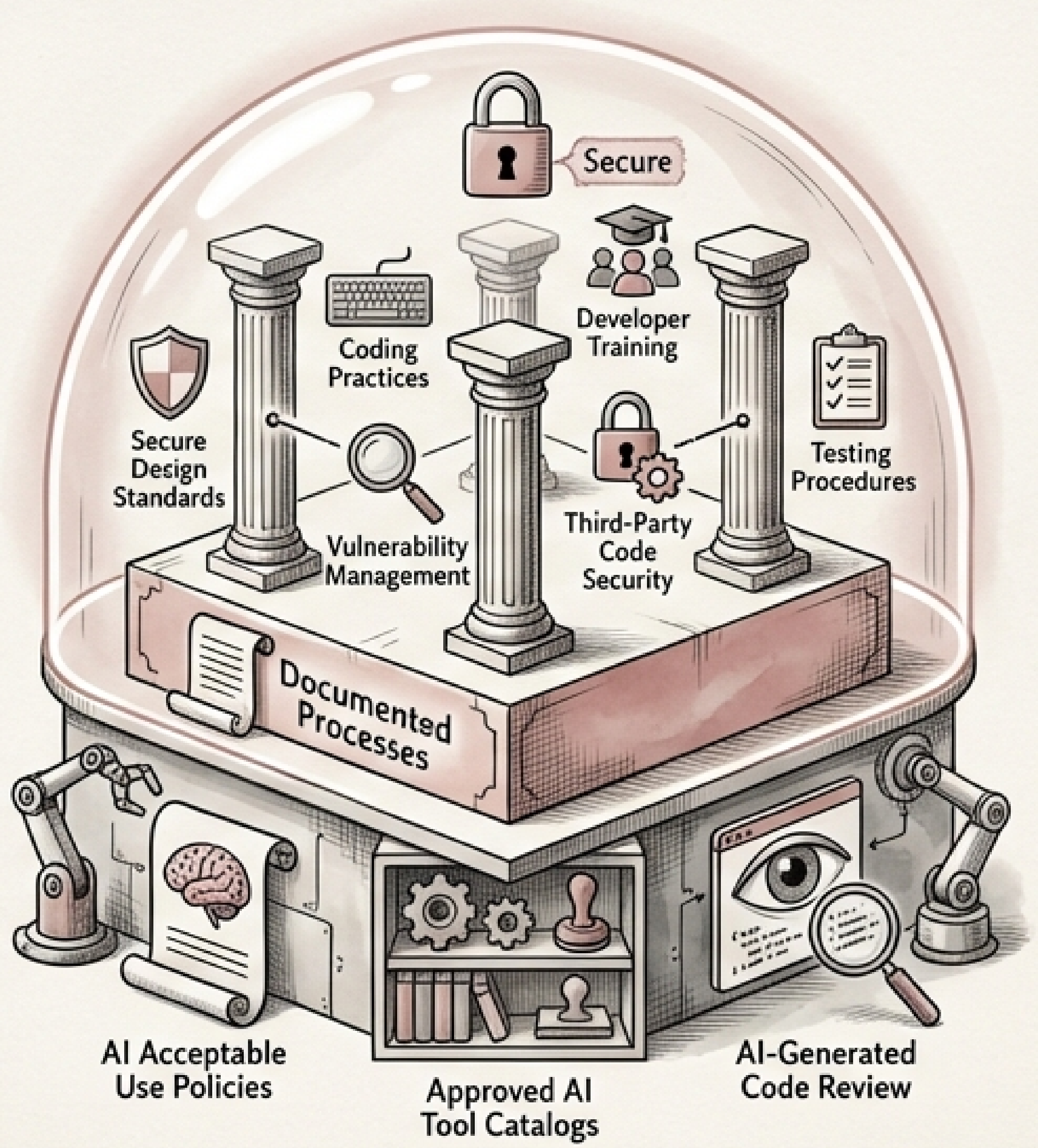


5. The goal is to enhance, not hinder, your team's velocity and innovation.



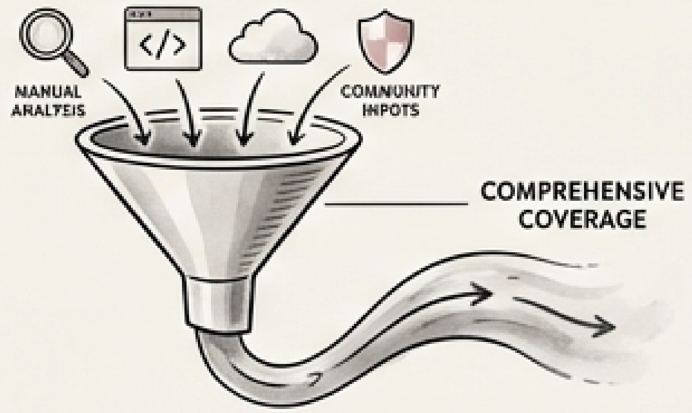
16.1: Establishing Your Secure Development Process - The Foundation

- The foundation of secure development lies in documented processes.
- Key processes include secure design standards, coding practices, and developer training.
- Also critical: vulnerability management, third-party code security, and testing procedures.
- For AI-augmented teams, incorporate AI acceptable use policies and approved AI tool catalogs.
- Mandate AI-generated code review requirements to catch potential issues.

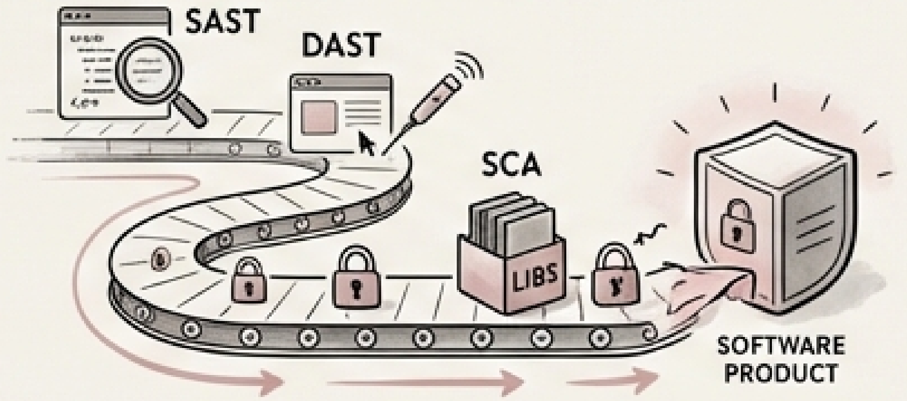


Comprehensive Vulnerability Intake: Sources and Prioritization

- Vulnerability information should be gathered from multiple sources for comprehensive coverage.

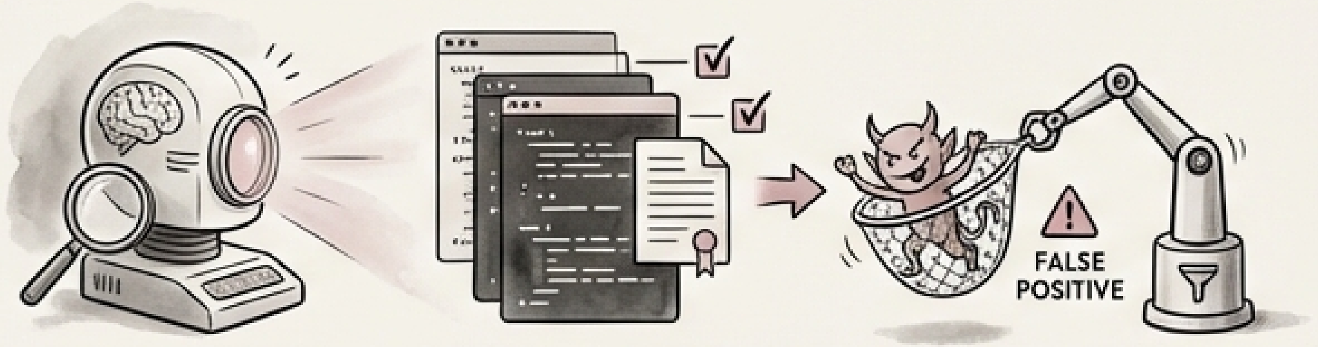


- Utilize SAST, DAST, and SCA tools to identify vulnerabilities early in the development lifecycle.

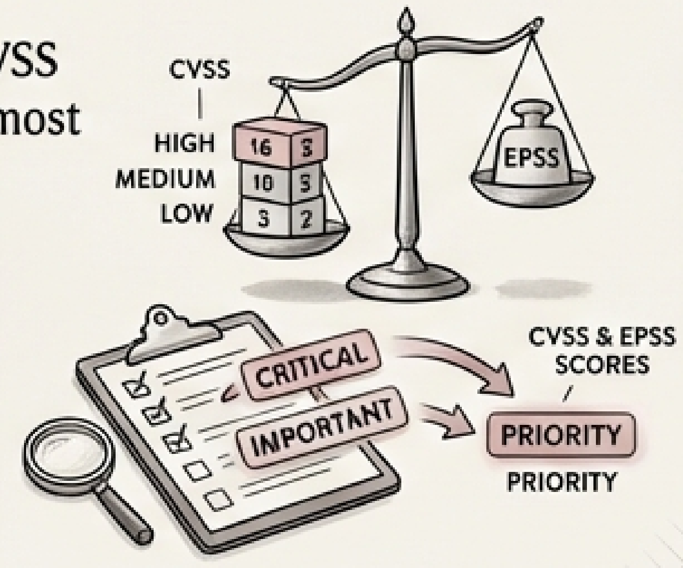


- Incorporate results from pen tests and bug bounty programs to uncover external attack vectors.

- Explore AI-assisted scanning tools to accelerate vulnerability detection, but be aware of false positives.

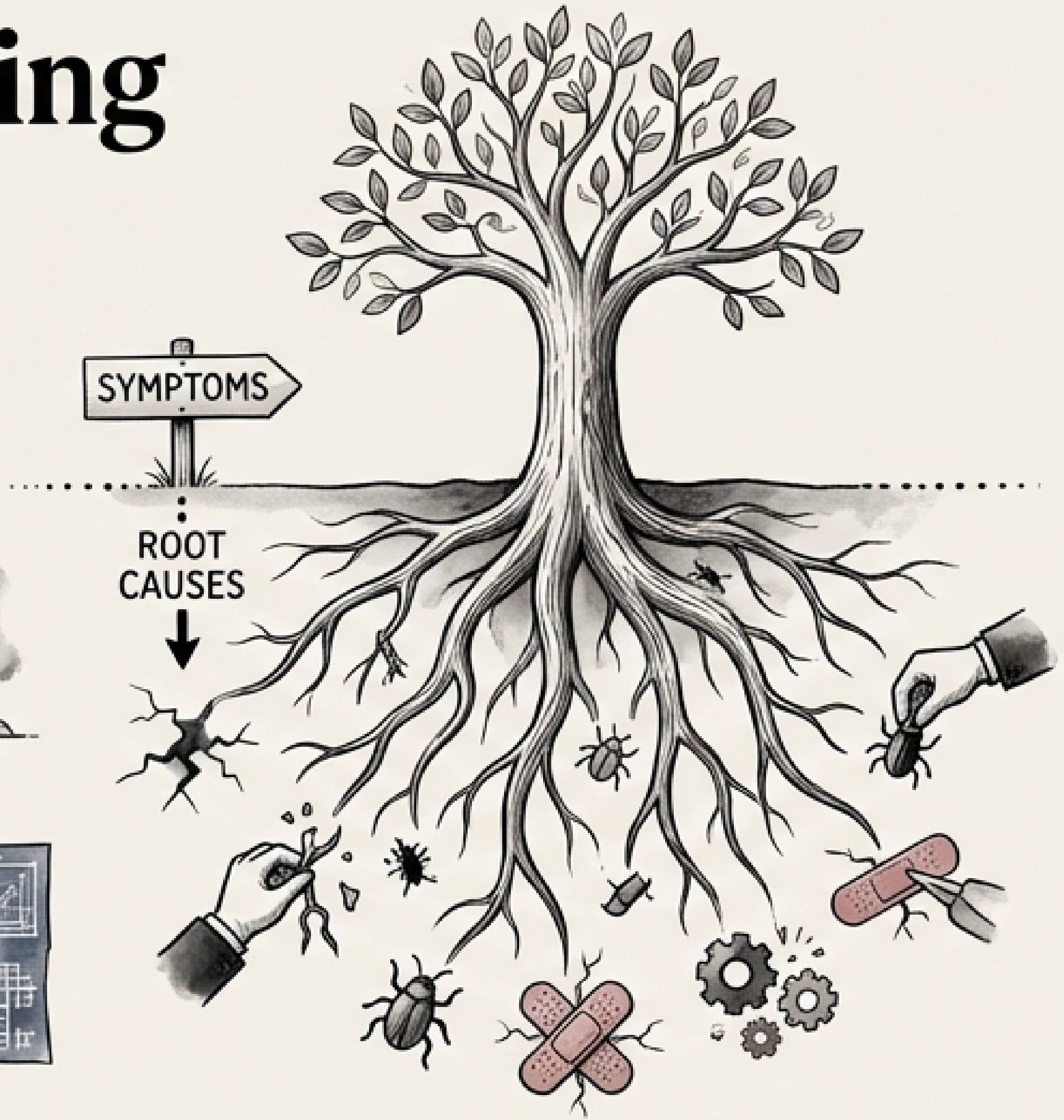


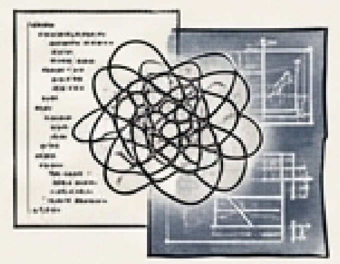
- Prioritize vulnerabilities using CVSS and EPSS scores to focus on the most critical issues.

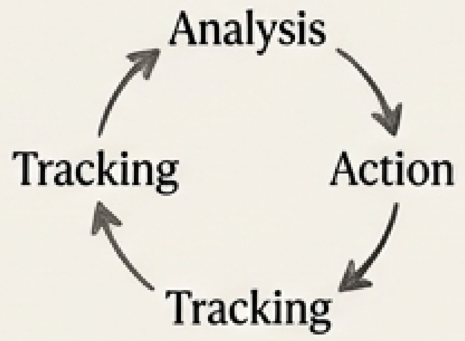


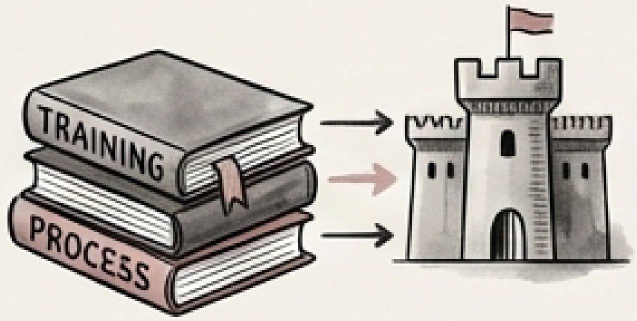
Root Cause Analysis: Preventing Recurring Vulnerabilities

- Address vulnerabilities with a focus on root cause analysis to prevent future occurrences.
- Simply patching the symptom without addressing the root cause is insufficient.
- Investigate the underlying reasons for vulnerabilities, such as coding errors or design flaws.
- Implement corrective actions, such as training updates or process improvements.
- Track identified root causes and associated corrective actions.

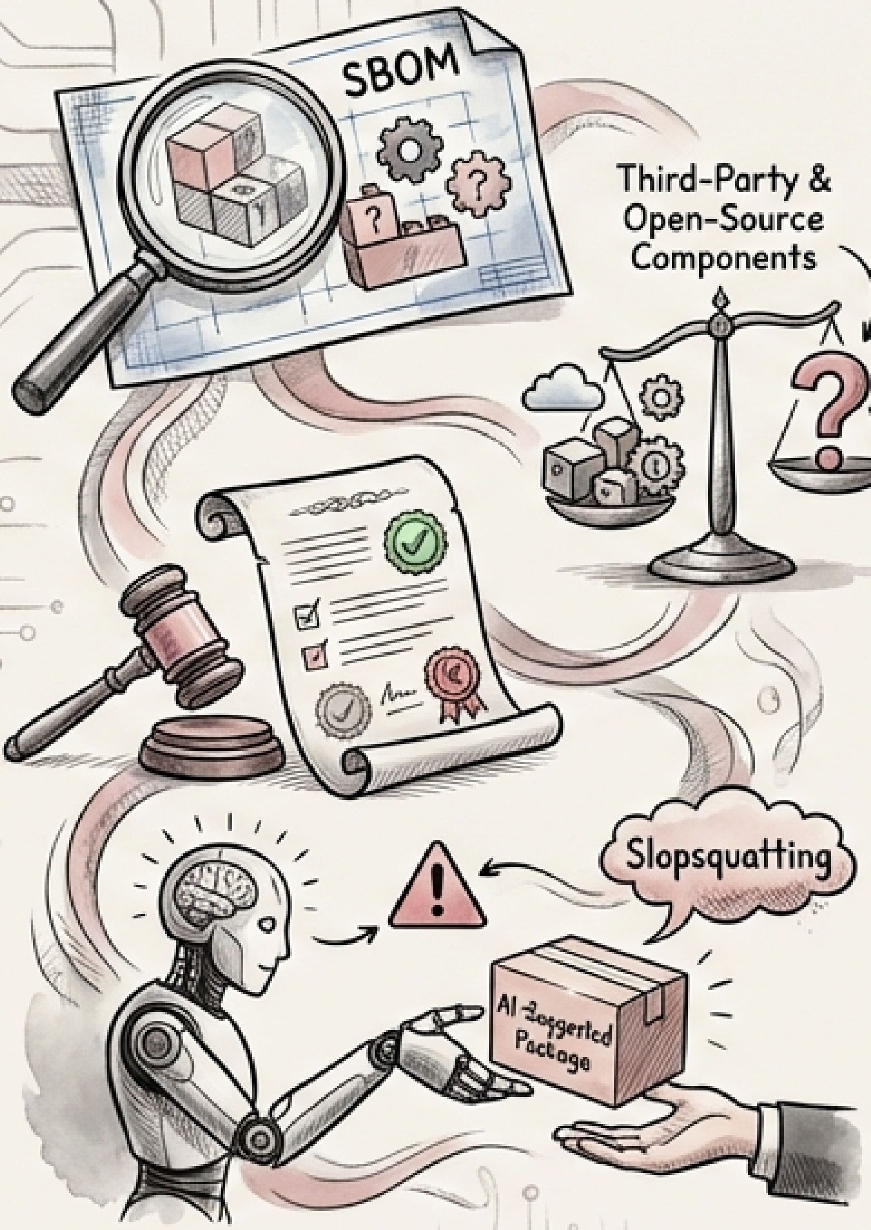








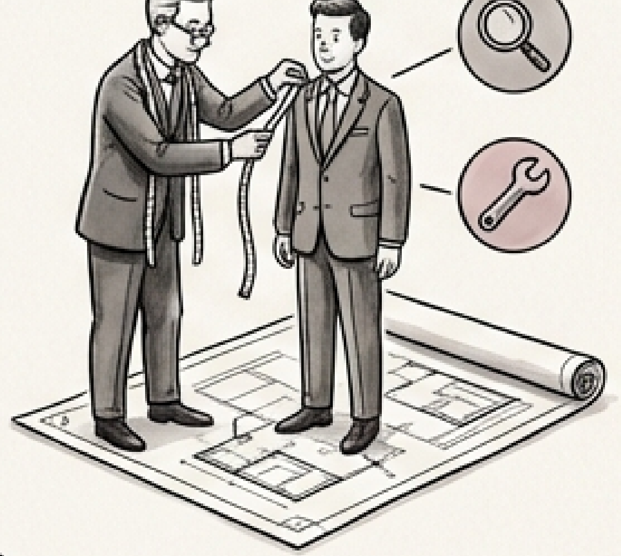
Software Composition Analysis (SCA): Knowing Your Dependencies



- Generate and maintain a Software Bill of Materials (SBOM) for complete dependency visibility.
- Conduct thorough risk assessments of third-party and open-source components.
- Implement license compliance scanning to avoid legal issues.
- AI teams must verify AI-suggested dependencies due to slopsquatting risks.
- 19.7% of AI package recommendations are non-existent, increasing security risk.

Secure Coding Training: Investing in Developer Skillsets

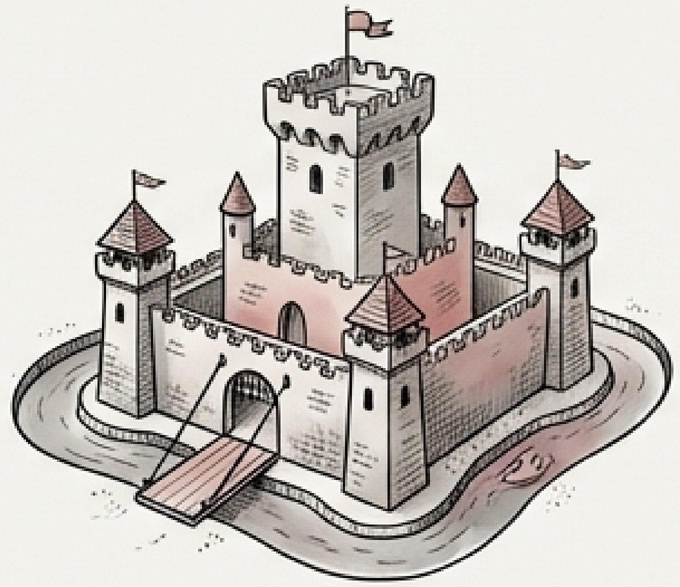
- Provide annual secure coding training for all development personnel.
- Tailor training to be environment-specific and role-specific for optimal relevance.
- General awareness training is not sufficient; focus on practical skills and techniques.
- Training should cover common vulnerabilities and how to prevent them in code.
- Include hands-on exercises and real-world examples to reinforce learning.



The Five Pillars of Secure Design

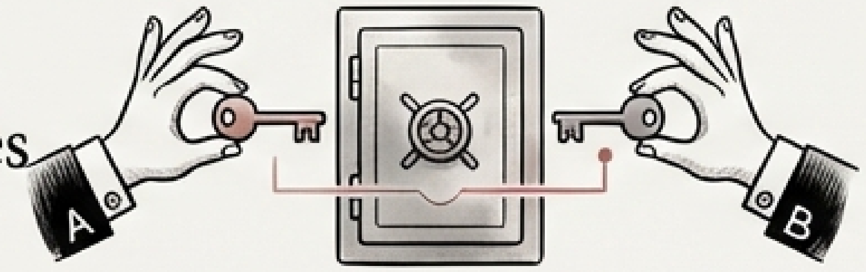
The five pillars of secure design provide a foundation for building resilient systems.

- 1. **Least Privilege:** Grant only the minimum necessary permissions.



- 2. **Defense in Depth:** Implement multiple layers of security controls.

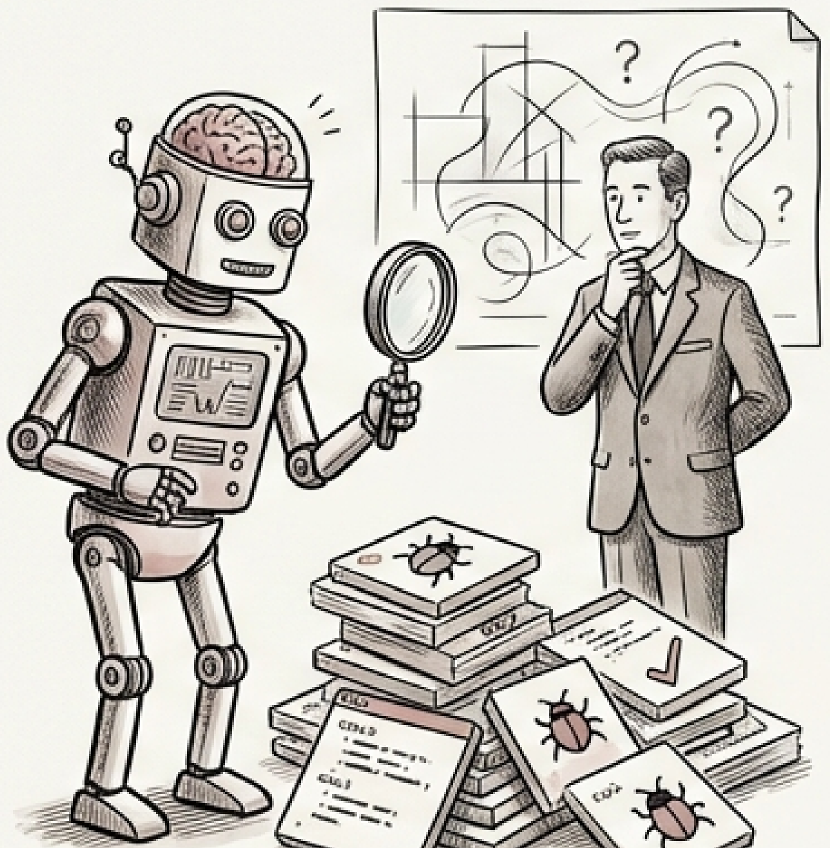
- 3. **Fail Secure:** Default to a secure state in case of failure.



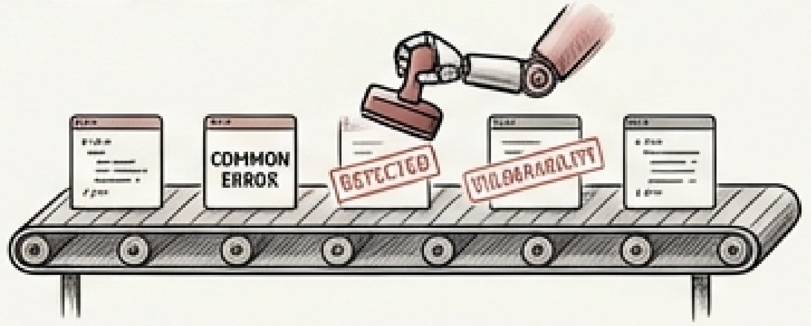
- 4. **Separation of Duties:** Divide responsibilities to prevent single points of failure.

AI-Assisted Code Review: Augmenting Human Expertise

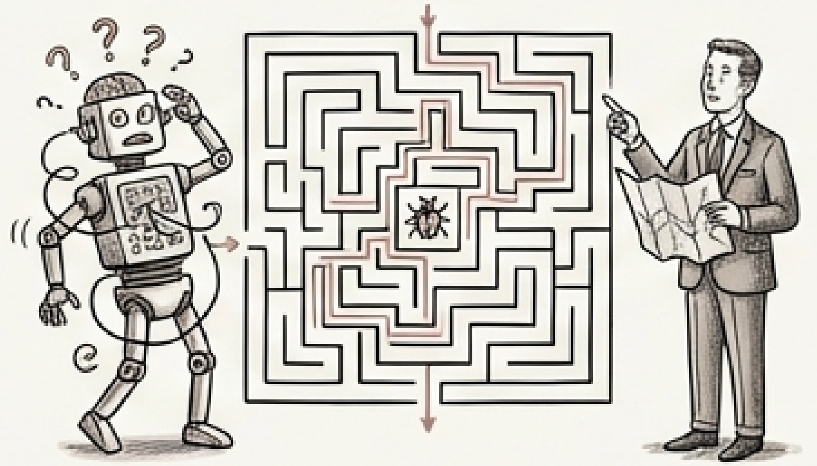
- AI code review tools can augment human judgment, but should not replace it entirely.



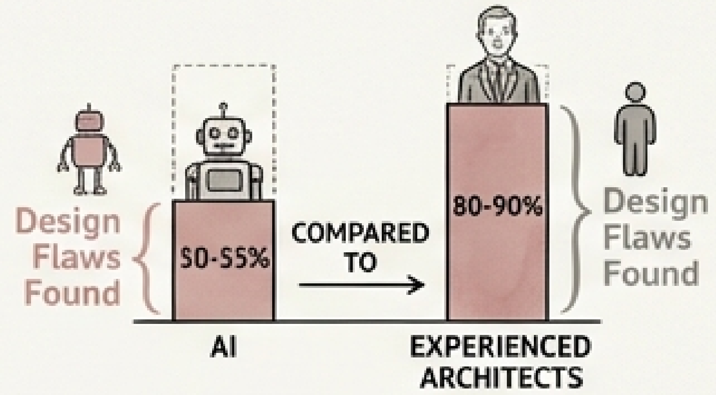
- AI can automate the detection of common coding errors and potential vulnerabilities.



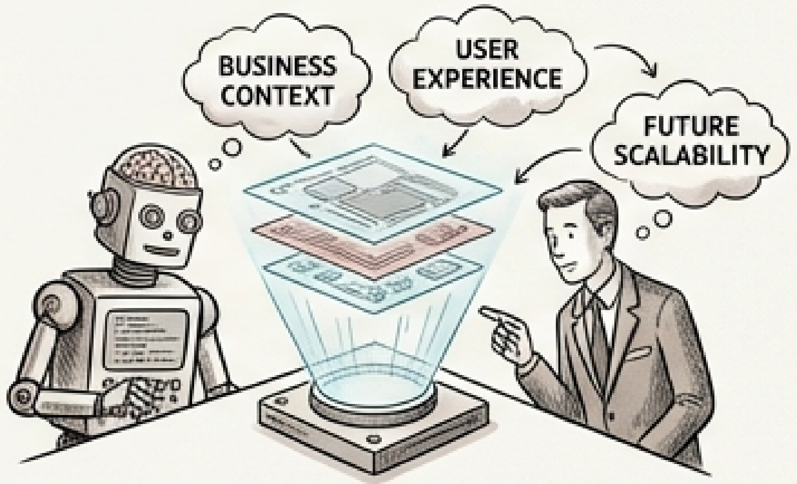
- However, AI may struggle with complex logic and nuanced design flaws.



- AI finds 50-55% of design flaws, compared to experienced architects.



- Human reviewers can provide valuable context and insights that AI may miss.



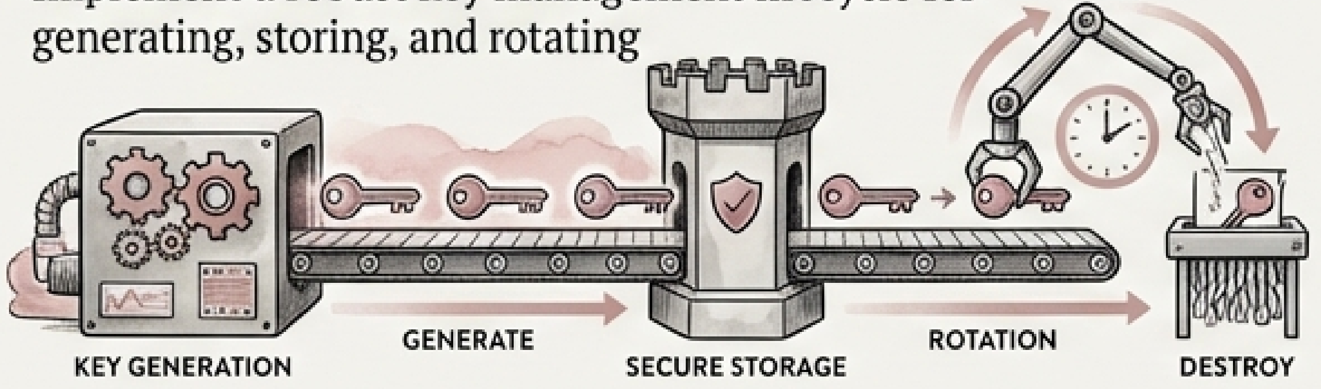
Enforcing Cryptographic Standards: Protecting Sensitive Data

STANDARDS & SECURITY

- Enforce strict cryptographic standards across all applications and systems.

- Use only approved cryptographic algorithms that have been rigorously vetted.

- Implement a robust key management lifecycle for generating, storing, and rotating

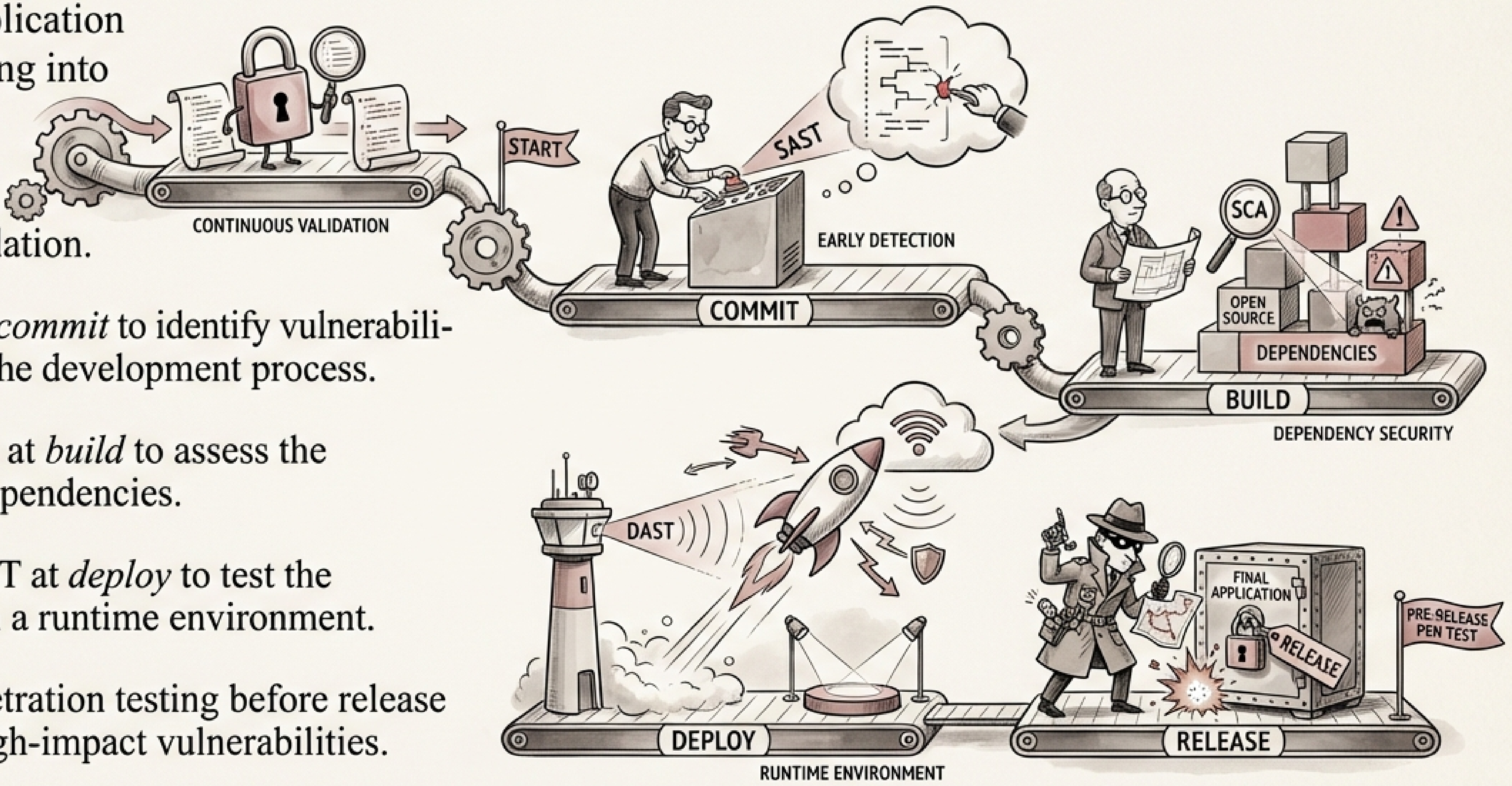


- Avoid custom cryptography, which is often vulnerable vulnerable to attacks.

- Protect cryptographic keys from unauthorized access and disclosure.

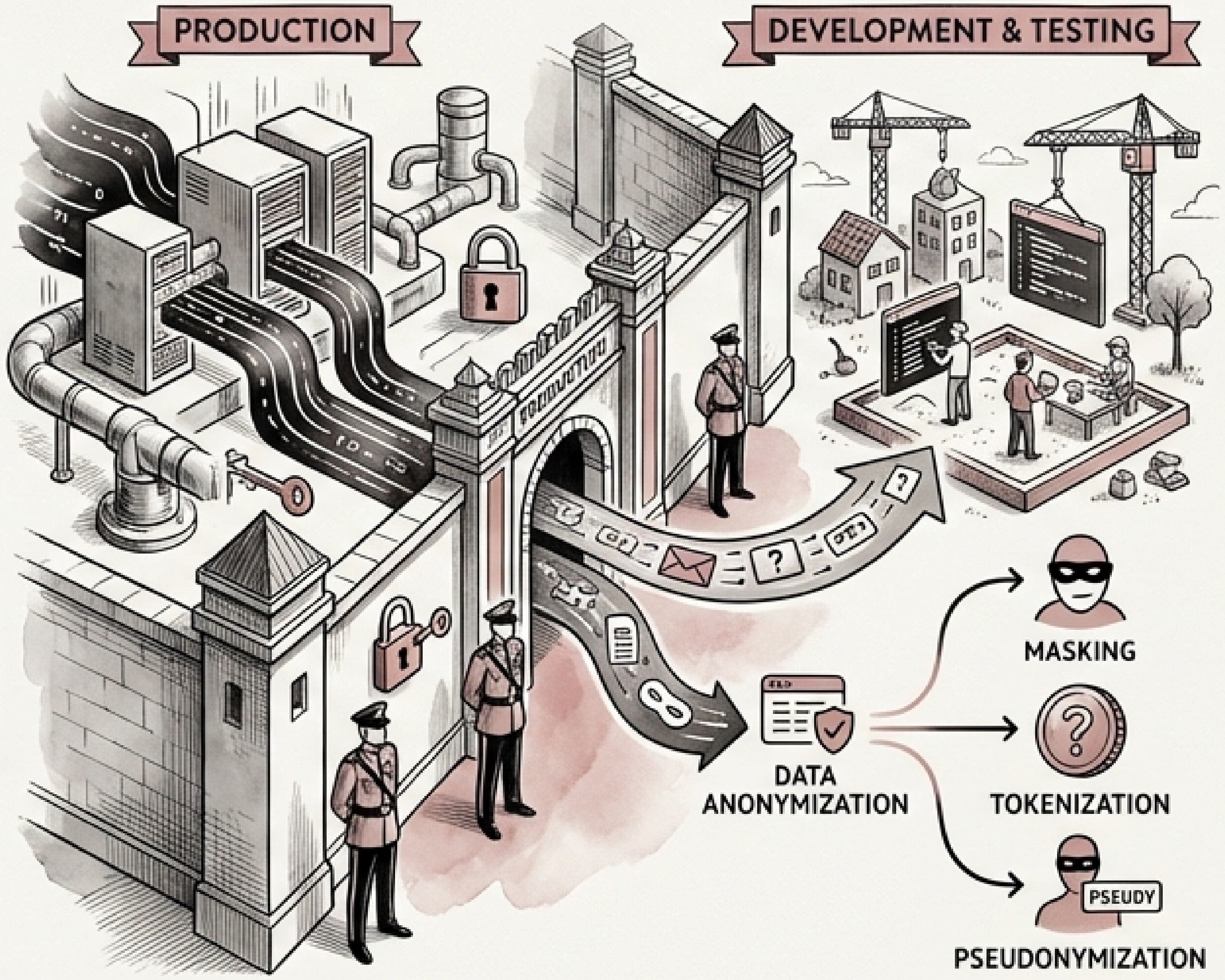
INTEGRATING SECURITY TESTING INTO CI/CD: SHIFT-LEFT SECURITY

- Integrate application security testing into the CI/CD pipeline for continuous security validation.
- Run SAST at *commit* to identify vulnerabilities early in the development process.
- Perform SCA at *build* to assess the security of dependencies.
- Execute DAST at *deploy* to test the application in a runtime environment.
- Conduct penetration testing before release to identify high-impact vulnerabilities.

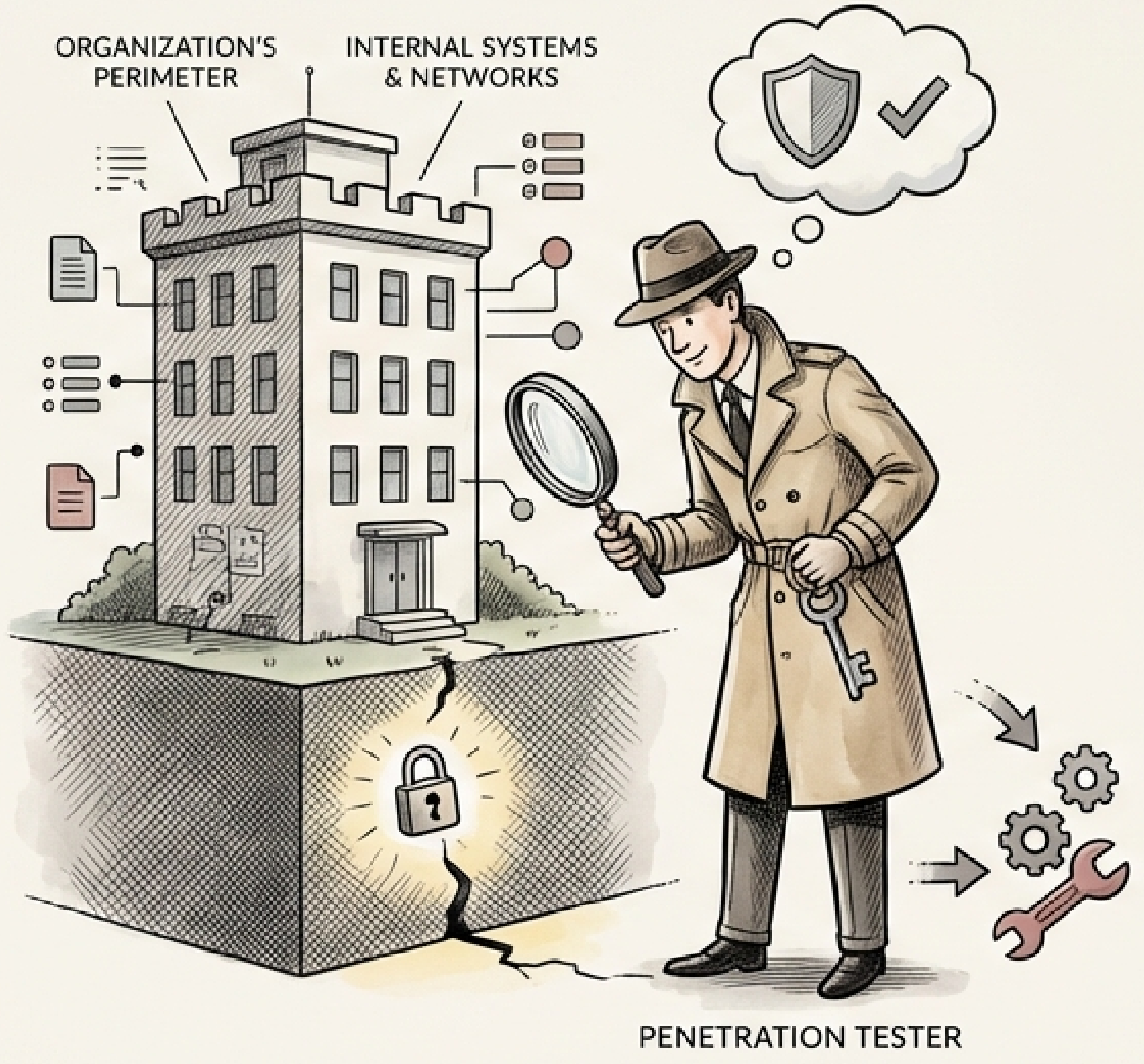


Environment Separation: Isolating Production Data

- Maintain separate production and non-production environments to minimize risk.
- Never use production data in development or testing environments without anonymization.
- Data anonymization techniques include masking, tokenization, and pseudonymization.
- Proper environment separation prevents accidental data breaches and unauthorized access.
- Implement strong access controls to restrict access to production environments.



PENETRATION TESTING: FINDING VULNERABILITIES BEFORE ATTACKERS DO



ESTABLISH A FORMAL PROGRAM:
Establish a formal penetration testing program to proactively identify security weaknesses.

ANNUAL TESTING:
Conduct both internal and external penetration tests at least annually.

INTERNAL ASSESSMENT:
Internal penetration tests assess the security of internal systems and networks.

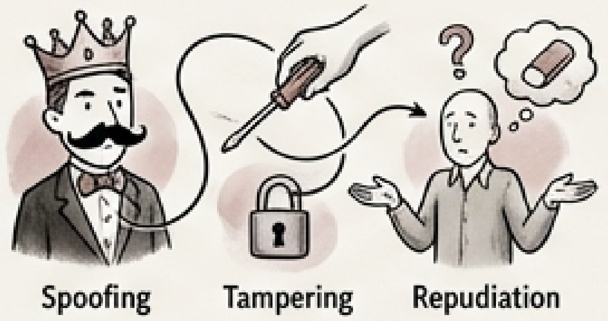
EXTERNAL SIMULATION:
External penetration tests simulate attacks from outside the organization's perimeter.

PRIORITIZE REMEDIATION:
Use the results of penetration tests to prioritize remediation efforts.

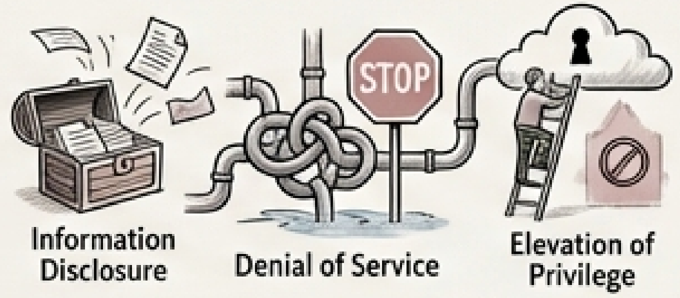
Threat Modeling: Proactively Identifying Security Risks

- Incorporate threat modeling into the design phase of software development.

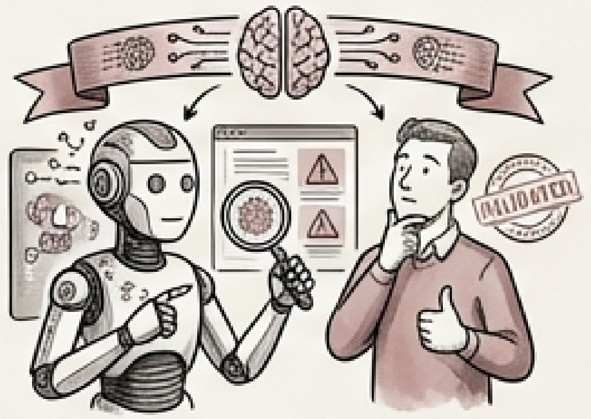
- Use threat modeling methodologies like STRIDE, PASTA, or attack trees to identify potential threats.



- STRIDE helps identify spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege risks.

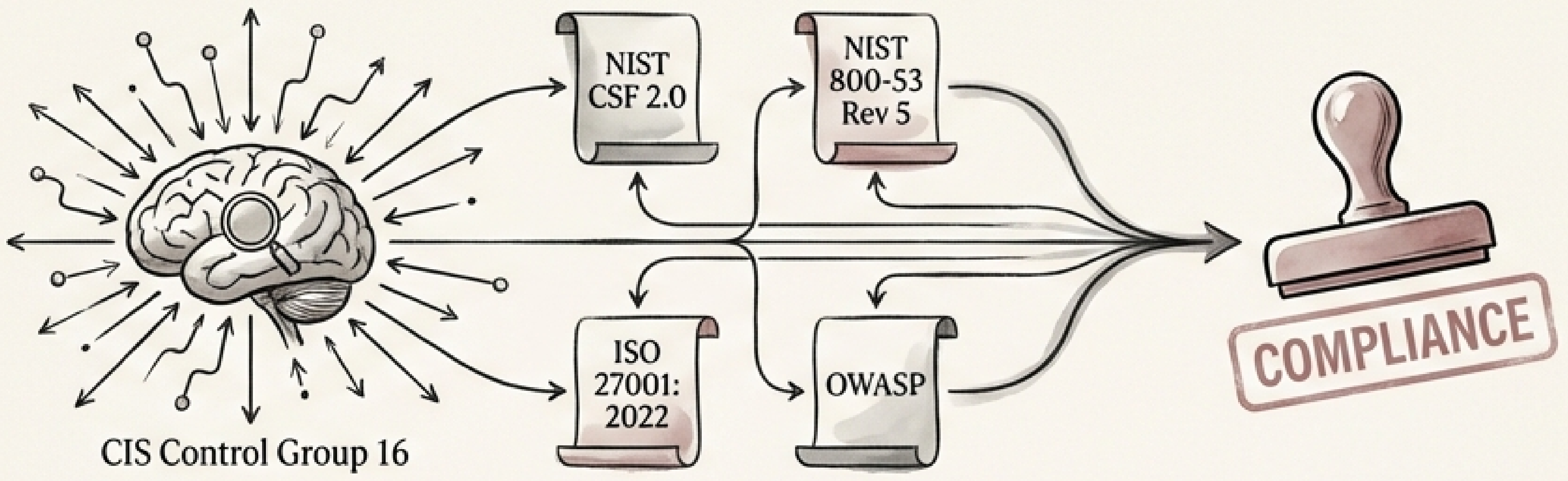


- AI-assisted threat modeling tools show promise, but require human validation.



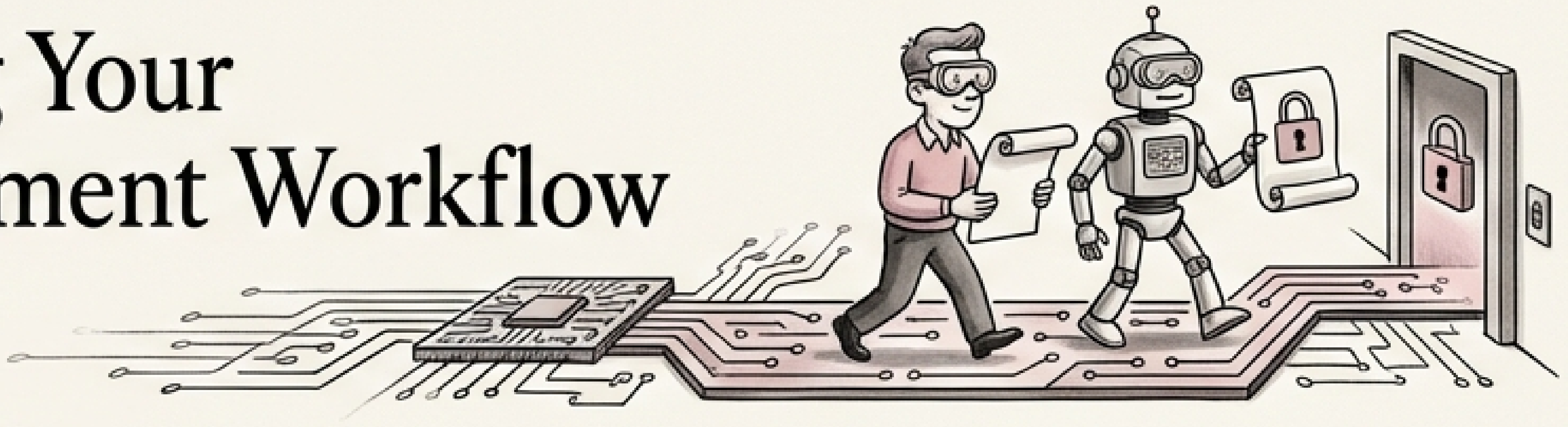
- Document identified threats and corresponding mitigation strategies.

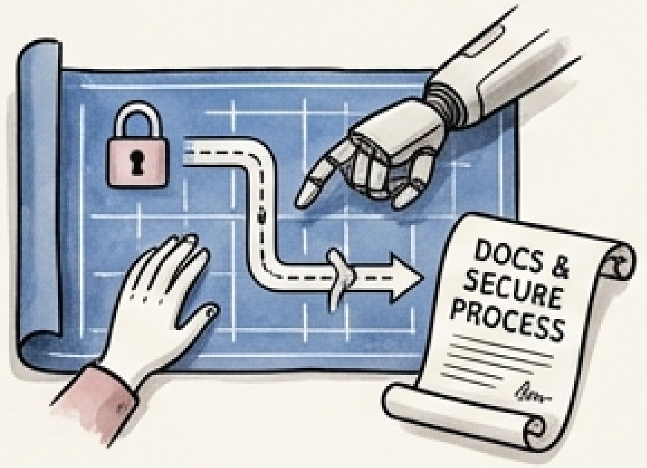
Cross-Framework Mapping: Demonstrating Compliance

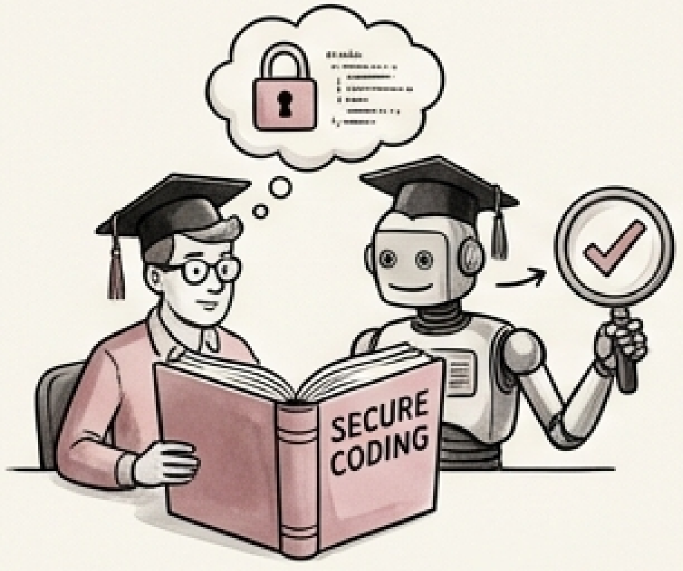


1. Each safeguard within CIS Control Group 16 maps to multiple other security frameworks.
2. These frameworks include NIST CSF 2.0, NIST 800-53 Rev 5, ISO 27001:2022, and OWASP.
3. This cross-mapping enables organizations to demonstrate compliance across multiple standards from a single implementation effort.
4. Reduces the burden of implementing and maintaining multiple sets of controls.
5. Improves efficiency and reduces costs associated with compliance.

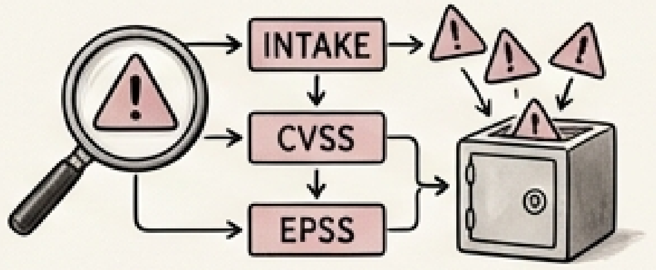
Key Takeaways: Securing Your AI-Augmented Development Workflow

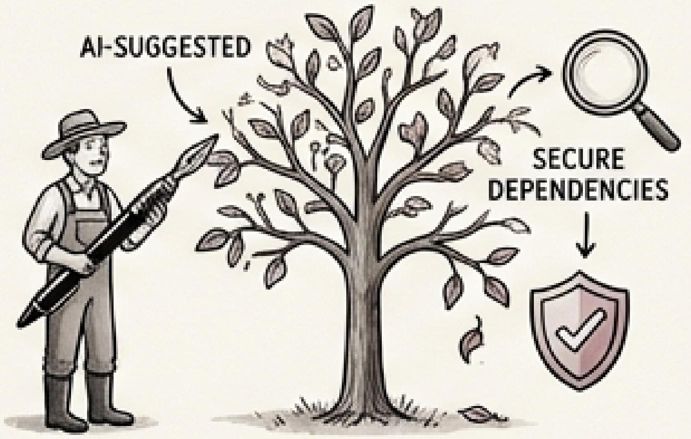


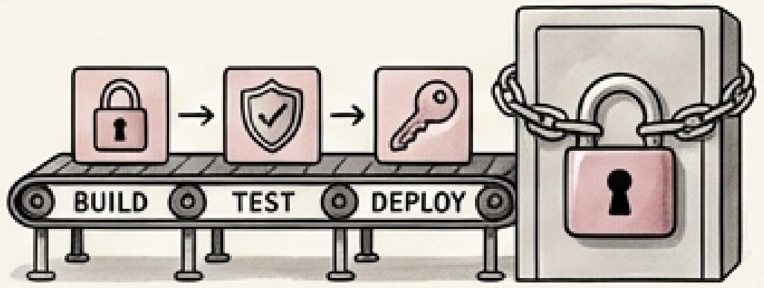




- Establish documented secure development processes tailored for AI-augmented teams.
- Implement comprehensive vulnerability intake and prioritization using CVSS + EPSS.
- Proactively manage software dependencies, especially AI-suggested packages.
- Invest in secure coding training and augment human review with AI tools.
- Integrate security testing into CI/CD and enforce cryptographic standards.







Thank You