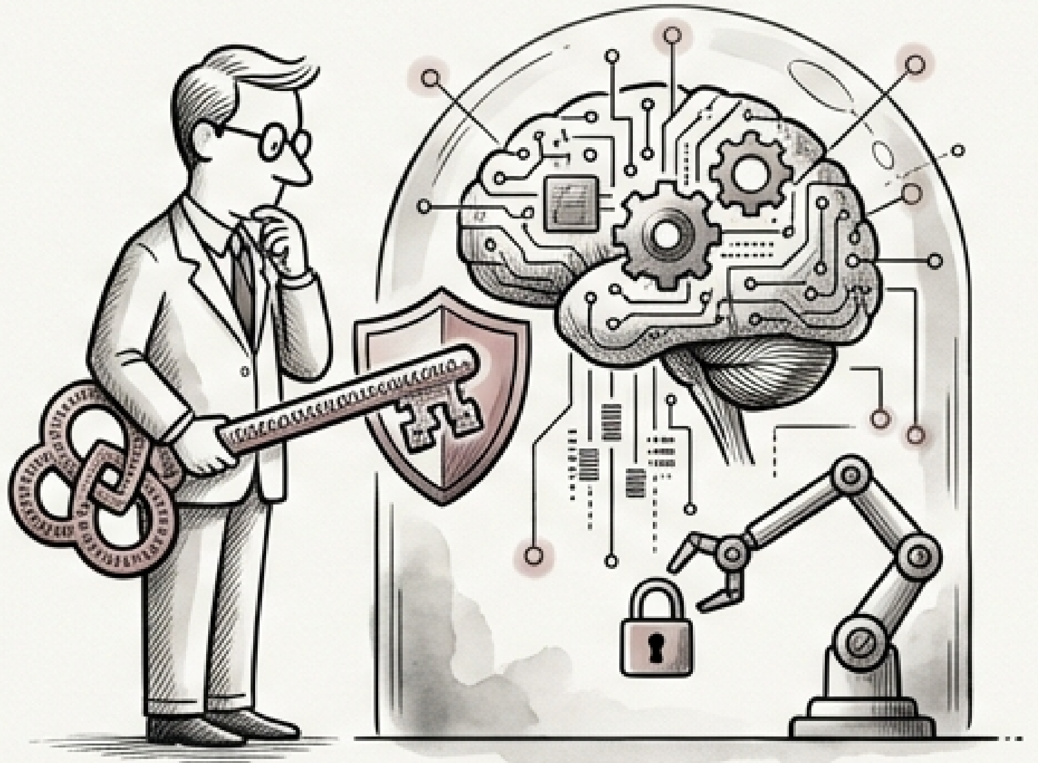
Securing AI-Augmented Systems: A Developer's Guide



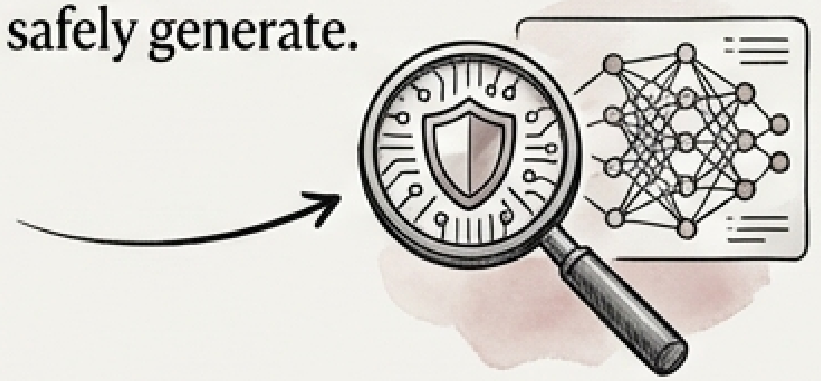
Securing AI-Augmented Systems: A Developer's Guide

☑ Security is most effectively addressed at the design phase, making it the most cost-effective investment.

```
<copy {
:  <secure
:    <code-al;
:      securecede);
:    </securey;
}</>
```

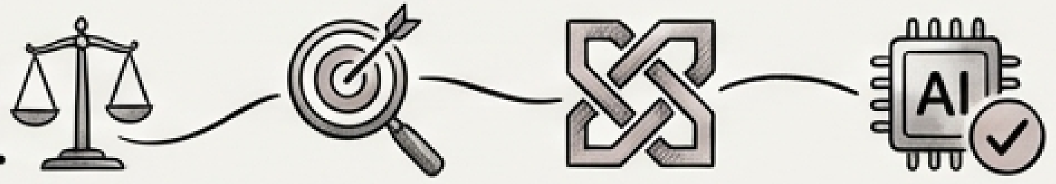
☑ A flawless implementation cannot compensate for an inherently insecure design.

☑ For AI-augmented teams, early design decisions significantly constrain the security boundaries of what AI tools can safely generate.

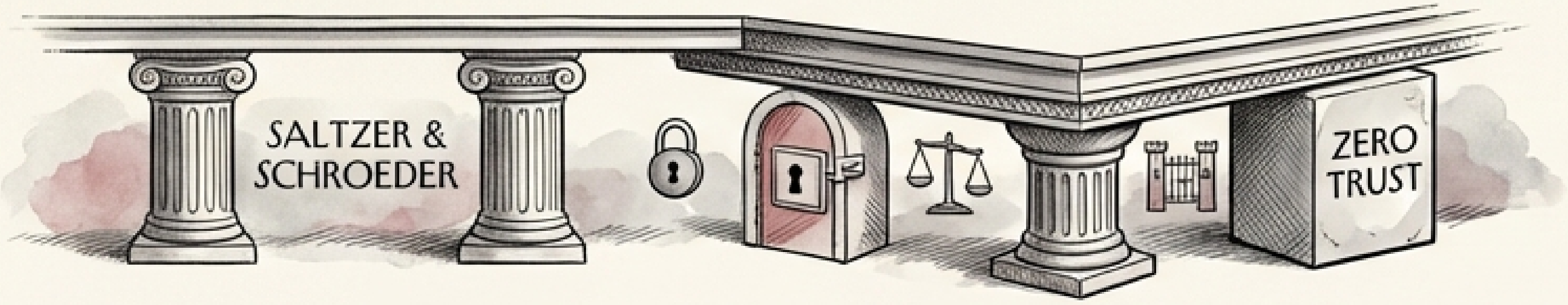


☑ This presentation focuses on applying secure design principles to AI-augmented systems and mitigating risks from AI-generated code.

☑ We'll cover key principles, attack surface analysis, secure design patterns, and AI architecture review best practices.



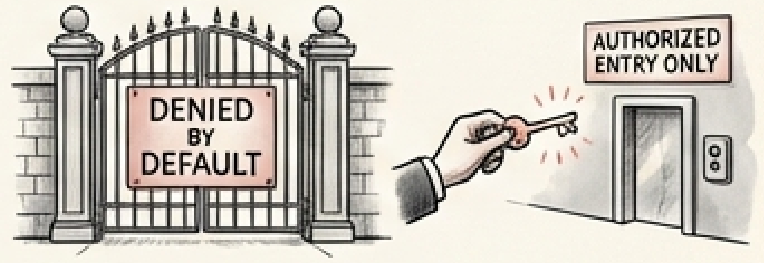
THE FOUNDATION: TEN SECURE DESIGN PRINCIPLES



- **Based on Saltzer/Schroeder principles plus Zero Trust:** *Economy of mechanism, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, psychological acceptability, defense in depth,* and *zero trust*.

- **Economy of mechanism:** Keep the design as simple and small as possible for easier verification and less attack surface.

- **Fail-safe defaults:** Access should be denied by default; grant access only when explicitly permitted.



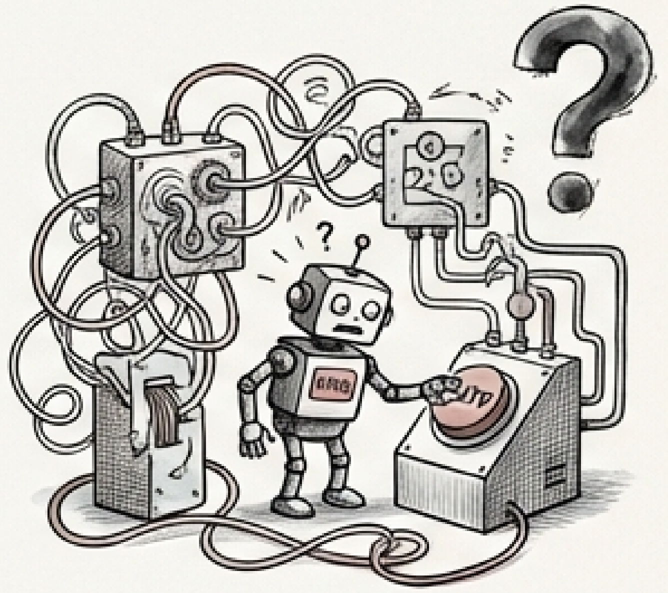
- **Complete mediation:** Every access to every object must be checked for authority.

- **Open design:** Security should not rely on secrecy; the design should be public and scrutinized.

Impact on AI-Generated Code: Economy of Mechanism & Open Design

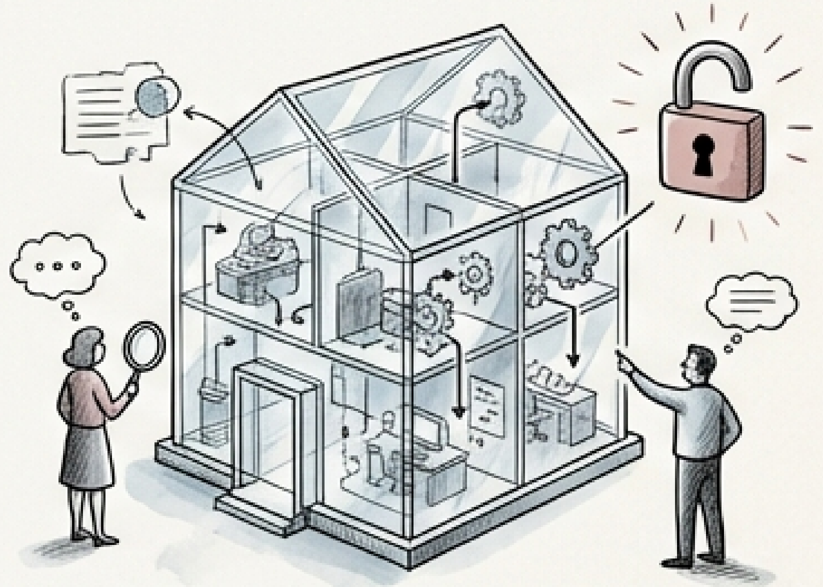
ECONOMY OF MECHANISM

- **Economy of Mechanism:** AI code generators can produce overly complex or verbose code. Demand concise, focused solutions in prompts.
- **Economy of Mechanism:** Refactor AI-generated code to remove unnecessary complexity and improve readability. This improves maintainability and reduces the likelihood of hidden vulnerabilities.



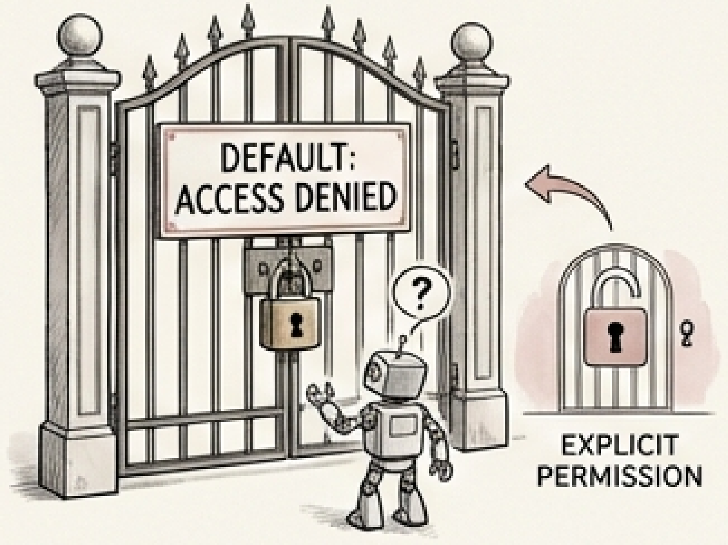
OPEN DESIGN

- **Open Design:** While AI code generators can obfuscate logic, ensure the overall architecture and interactions are transparent and well-documented.
- **Open Design:** Subject AI-generated code to peer review to identify potential security flaws that might be missed by automated tools.
- **Open Design:** Avoid relying on AI to generate proprietary security mechanisms that cannot be independently verified.

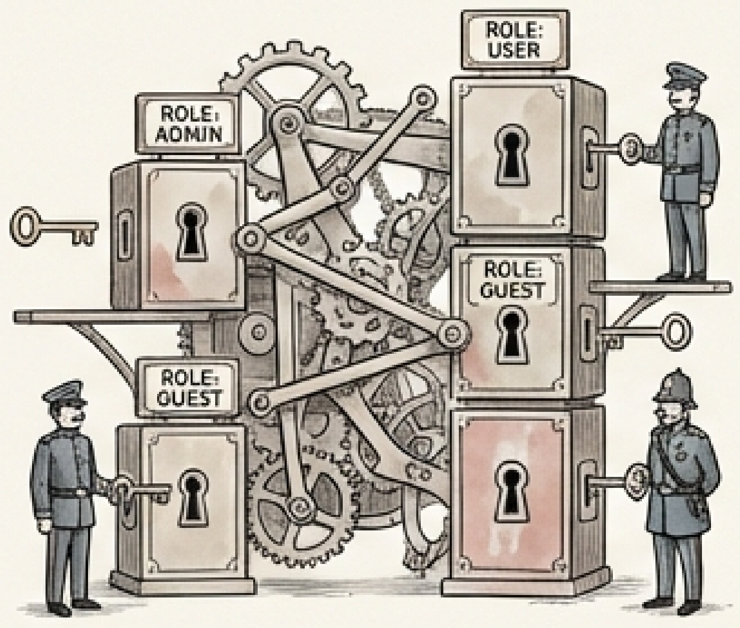


Mitigating Risk: Fail-Safe Defaults & Complete Mediation

The New Yorker Editorial Style: Intelligent Visual Metaphors for Secure Systems

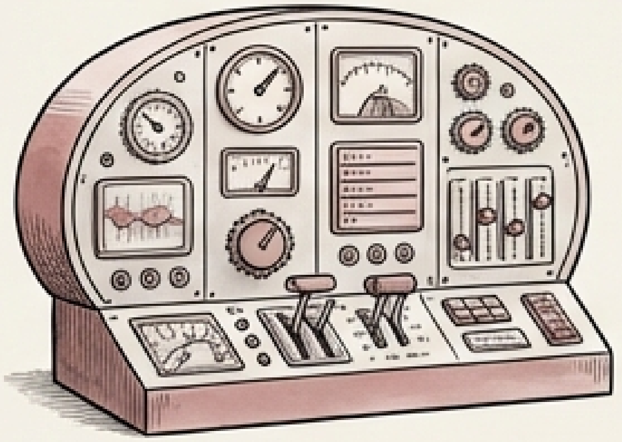


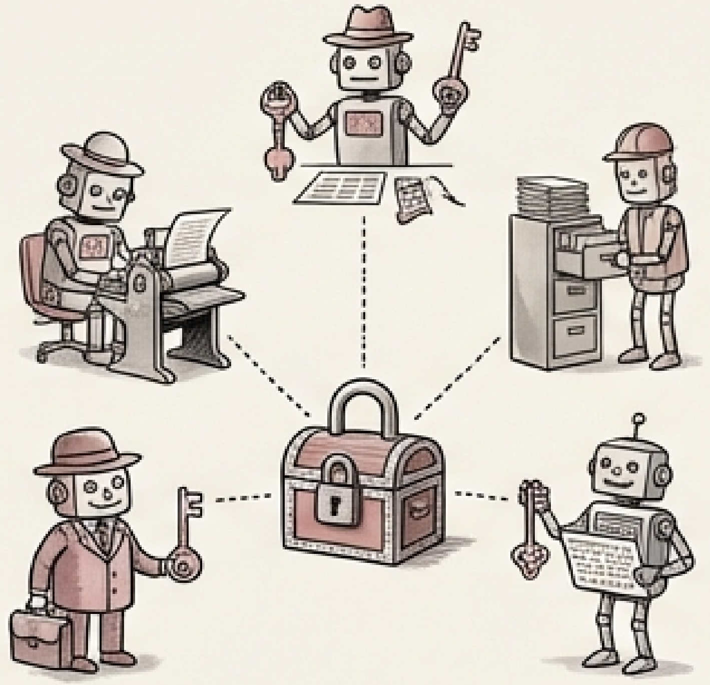
- **Fail-Safe Defaults:** AI-generated code should default to denying access unless explicitly permitted through defined access control policies.
- **Fail-Safe Defaults:** Configure AI tools to have minimal necessary permissions by default, limiting their potential impact in case of compromise.
- **Complete Mediation:** All requests to access resources, especially those initiated by AI tools or agents, must be explicitly authorized.
- **Complete Mediation:** Implement robust access control mechanisms (e.g., role-based access control) and enforce them consistently.
- **Complete Mediation:** Log all access attempts and authorization decisions for auditing and forensic analysis.



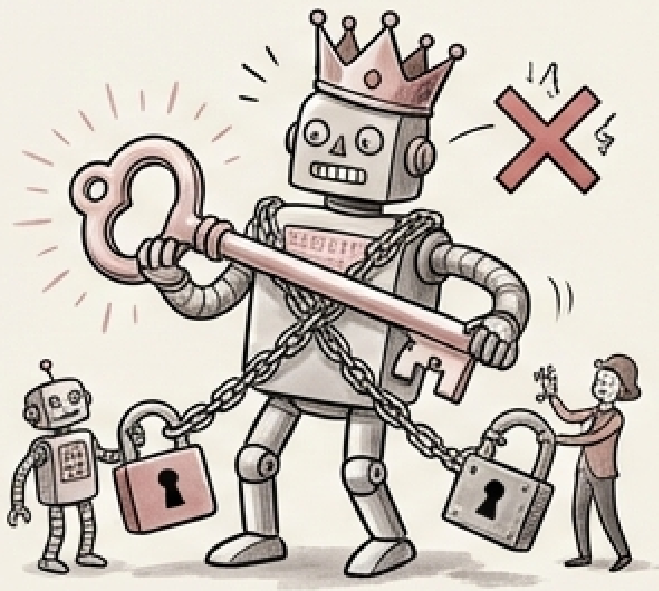
Privilege Control: Separation of Privilege & Least Privilege

SEPARATION OF PRIVILEGE

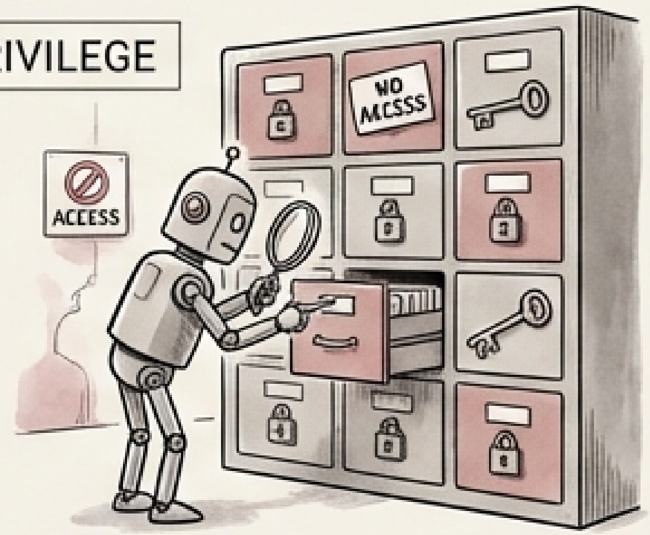




- Divide AI tasks and responsibilities among different AI tools or agents, each with specific permissions.
- Avoid granting any single AI tool or agent complete control over the entire system or sensitive data.



LEAST PRIVILEGE

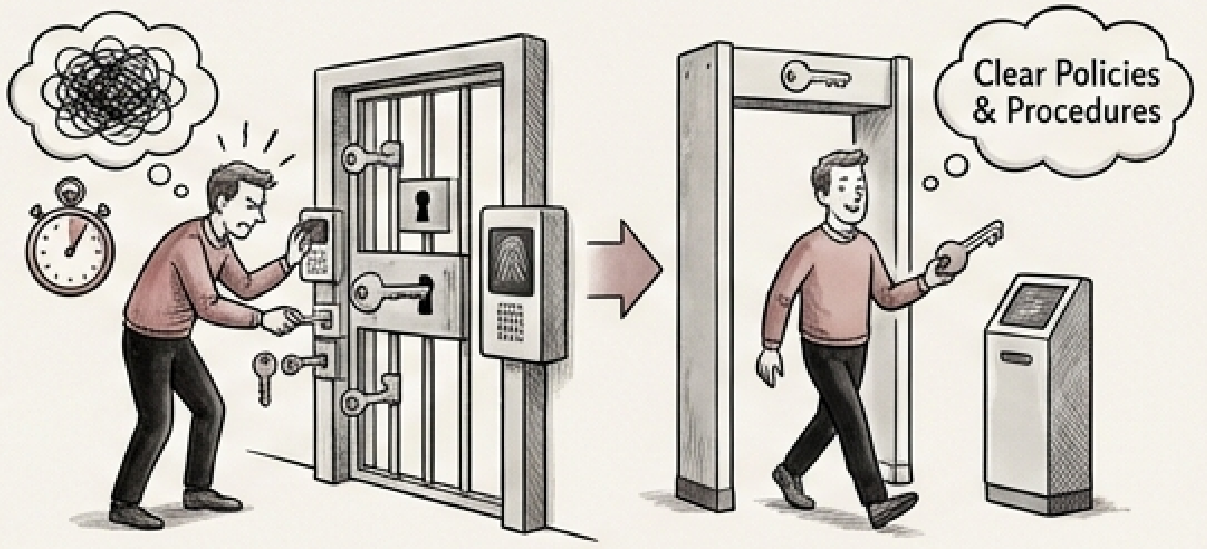


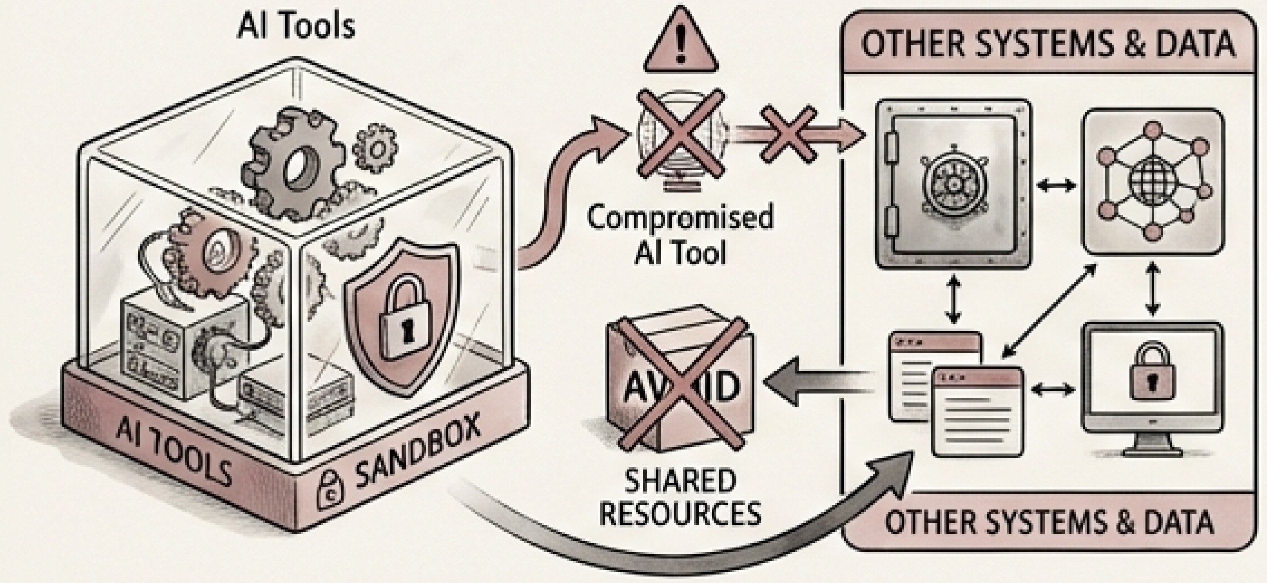
- Grant AI tools and agents only the minimum necessary permissions to perform their intended functions.

- Regularly review and revoke permissions that are no longer required by AI tools or agents.
- Use temporary credentials for AI tools accessing sensitive data; rotate these credentials frequently.

USER EXPERIENCE AND SECURITY: PSYCHOLOGICAL ACCEPTABILITY & LEAST COMMON MECHANISM

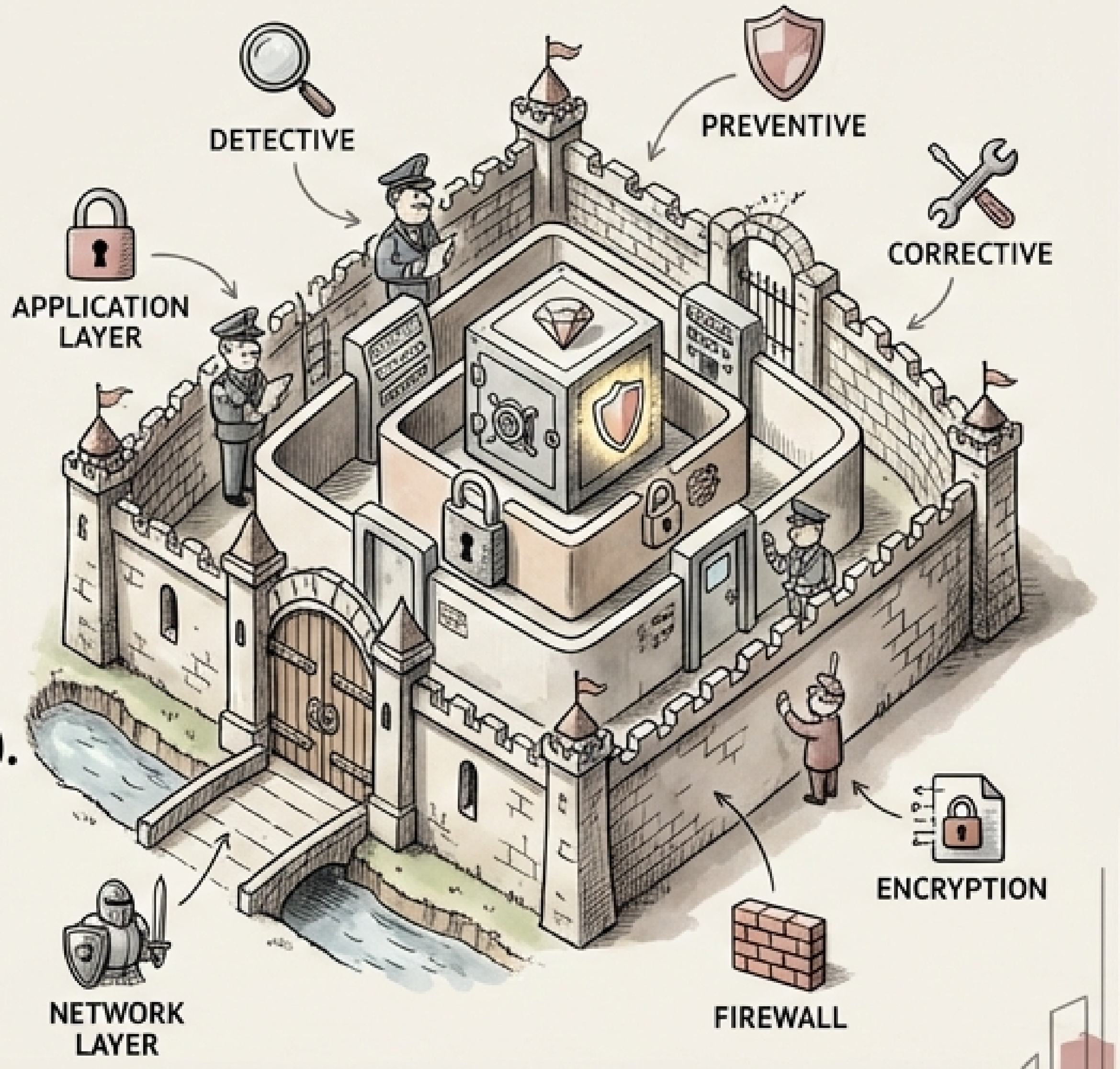
- **Psychological Acceptability:** Security measures should be user-friendly and not unduly burdensome to avoid user workarounds or security fatigue.
- **Psychological Acceptability:** Provide clear and concise explanations for security policies and procedures related to AI tools.
- **Least Common Mechanism:** Minimize shared components and resources between AI tools and other systems to reduce the risk of cascading failures or vulnerabilities.
- **Least Common Mechanism:** Avoid shared authentication or authorization mechanisms that could allow a compromised AI tool to access unrelated systems.
- **Least Common Mechanism:** Isolate AI tools in sandboxed environments to prevent them from interfering with other applications or data.





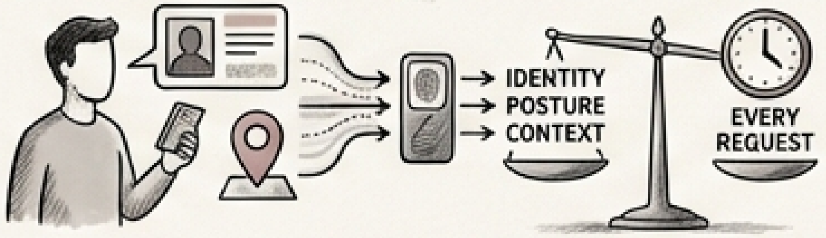
DEFENSE IN DEPTH: LAYERING SECURITY CONTROLS

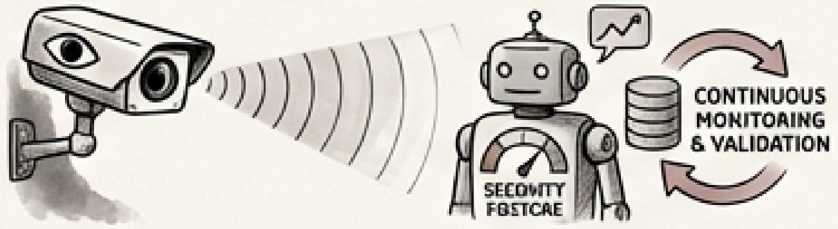
- **Defense in Depth:** Implement multiple layers of security controls, so that a failure in one layer does not compromise the entire system.
- **Defense in Depth:** Use a combination of preventive, detective, and corrective security measures.
- **Defense in Depth:** Apply security controls at different levels, including network, host, application, and data layers.
- **Defense in Depth:** Example: Input validation (application), firewall (network), intrusion detection (host), encryption (data).
- **Defense in Depth:** Regularly test and evaluate the effectiveness of each security layer.

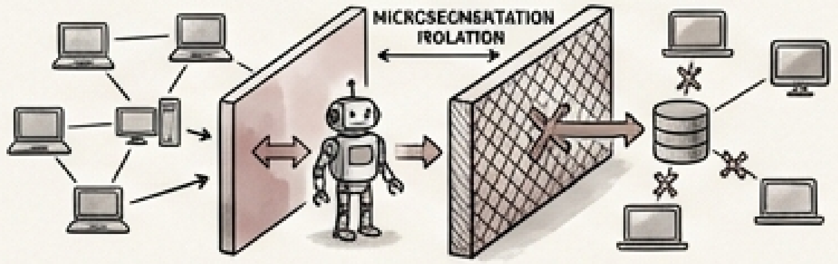


Zero Trust: Verify Everything, Trust Nothing

- Zero Trust: Never automatically trust any user, device, or application, regardless of location or network.
- Zero Trust: Always verify the identity, posture, and context of every access request.
- Zero Trust: Use multi-factor authentication (MFA) for all users and AI tools accessing sensitive resources.
- Zero Trust: Continuously monitor and validate the security posture of AI tools and agents.
- Zero Trust: Implement microsegmentation to isolate AI tools and limit their lateral movement in the network.

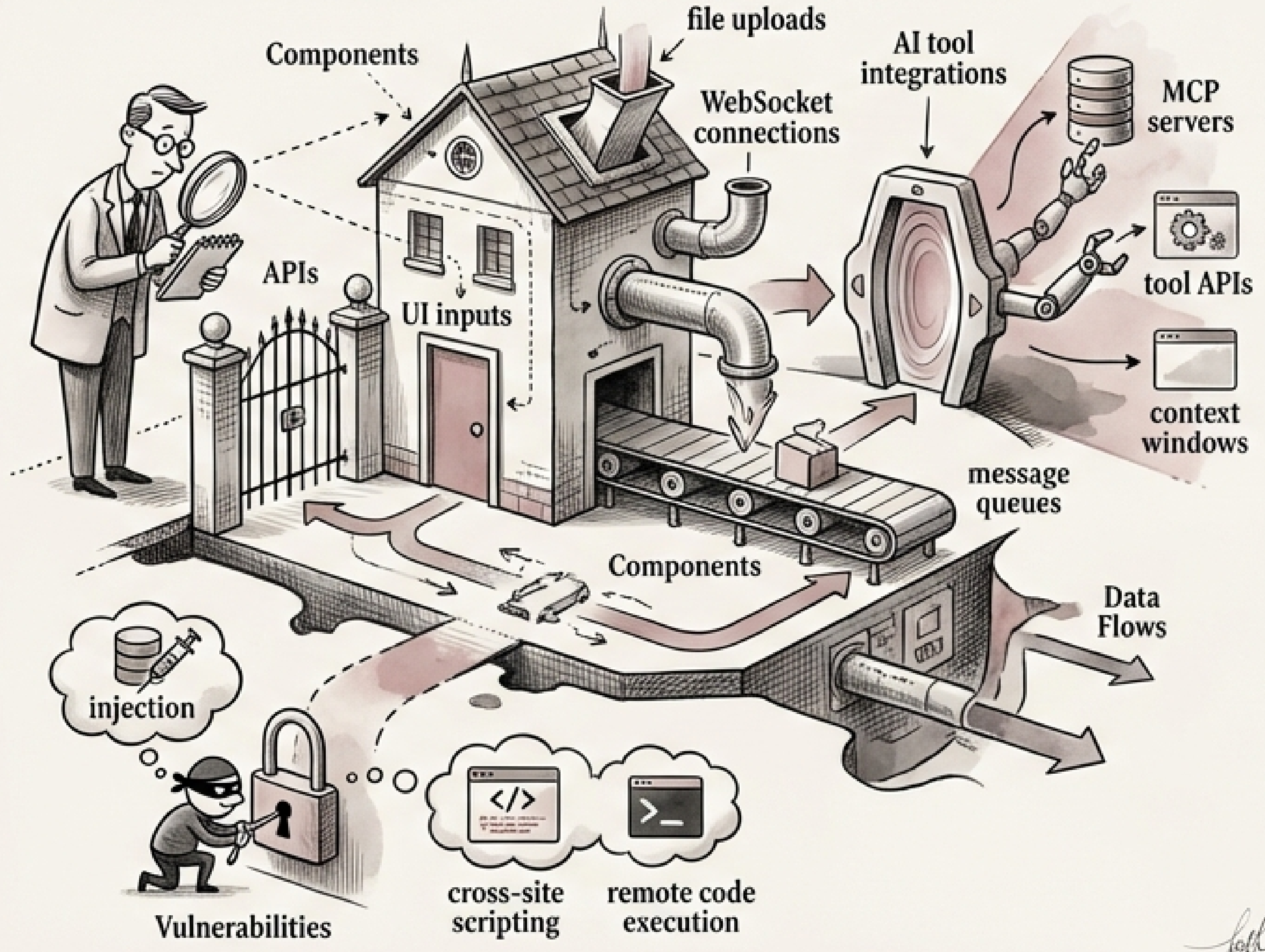






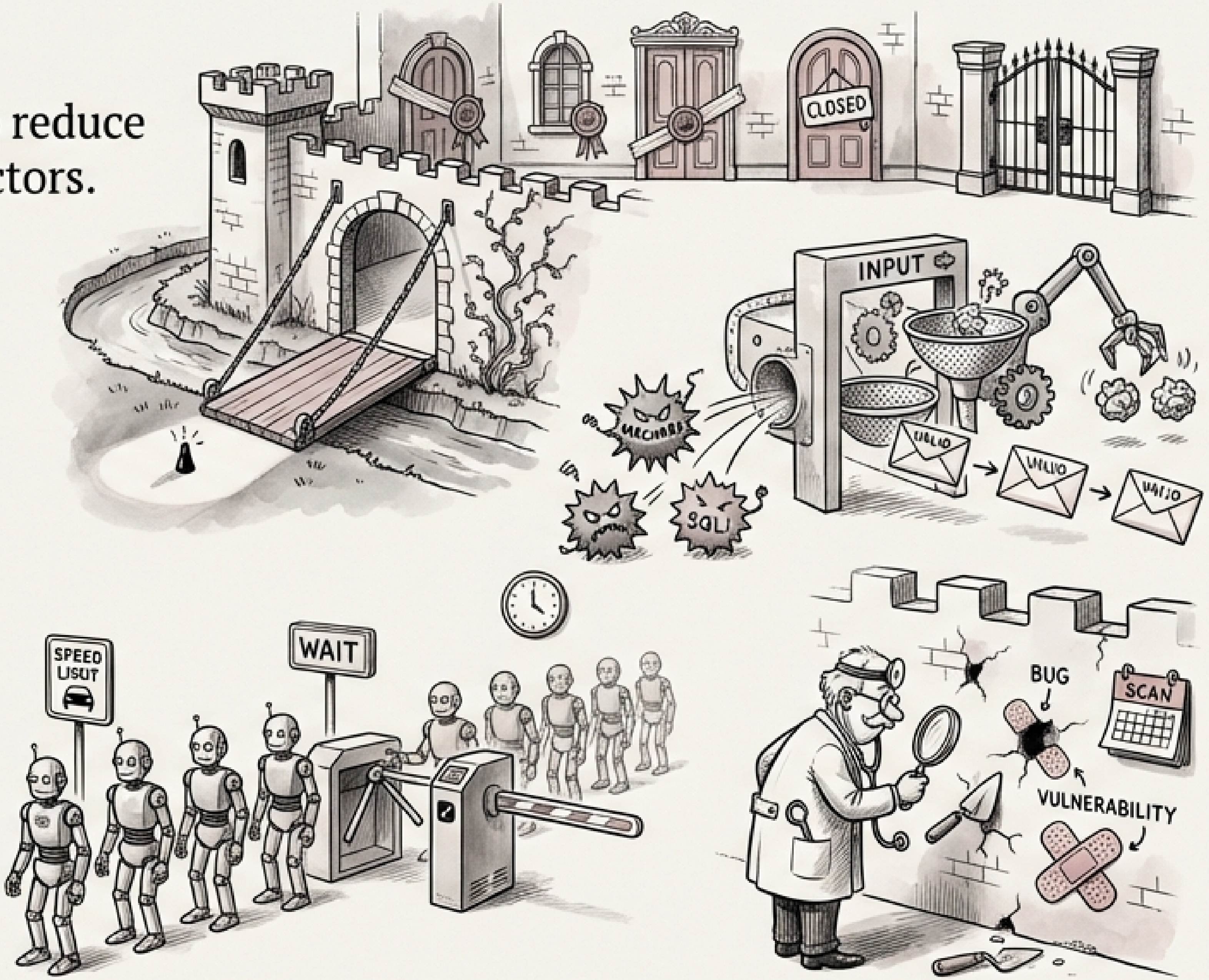
Attack Surface Analysis: Identifying Entry Points

- **Attack Surface Analysis:** Enumerate all potential entry points where attackers could gain access to the system.
- Key entry points include: APIs, UI inputs, file uploads, WebSocket connections, message queues, and AI tool integrations.
- AI tools significantly increase the attack surface, introducing new entry points like MCP servers, tool APIs, and context windows.
- Carefully document all data flows and interactions between different components of the system.
- Identify potential vulnerabilities in each entry point, such as SQL injection, cross-site scripting, or remote code execution.

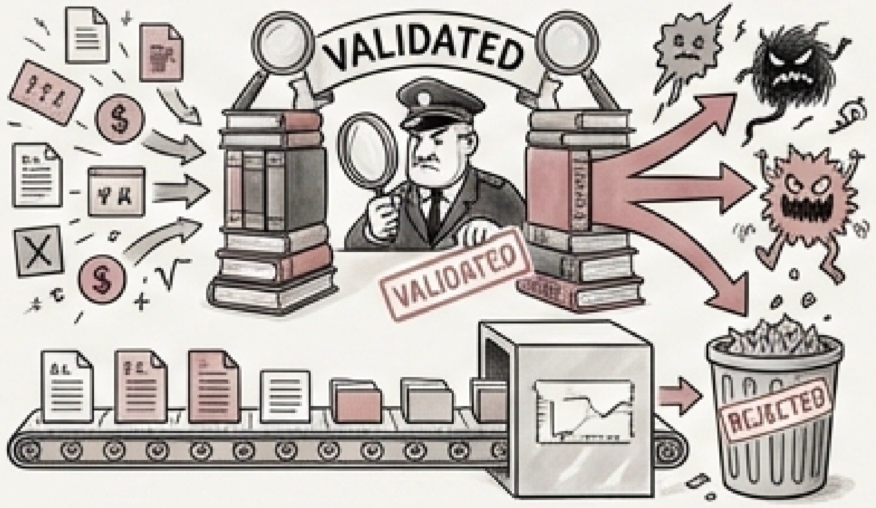


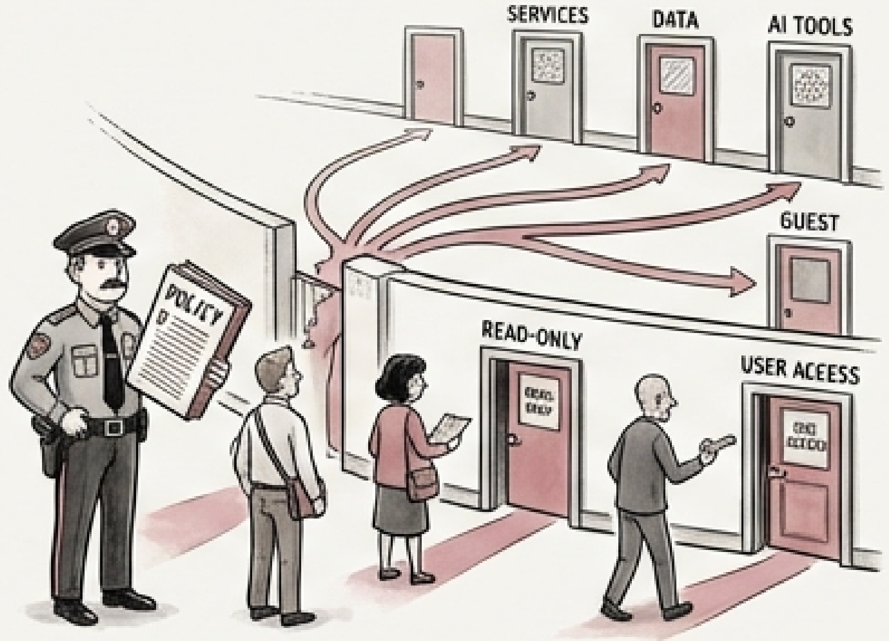
Attack Surface Reduction: Minimizing Exposure

- Disable unused endpoints and features to reduce reduce the number of potential attack vectors.
- Minimize the amount of exposed functionality to only what is strictly necessary.
- Enforce strict input validation schemas to prevent malicious data from entering the system.
- Implement rate limiting to prevent denial-of-service attacks.
- Regularly scan for vulnerabilities and patch them promptly.

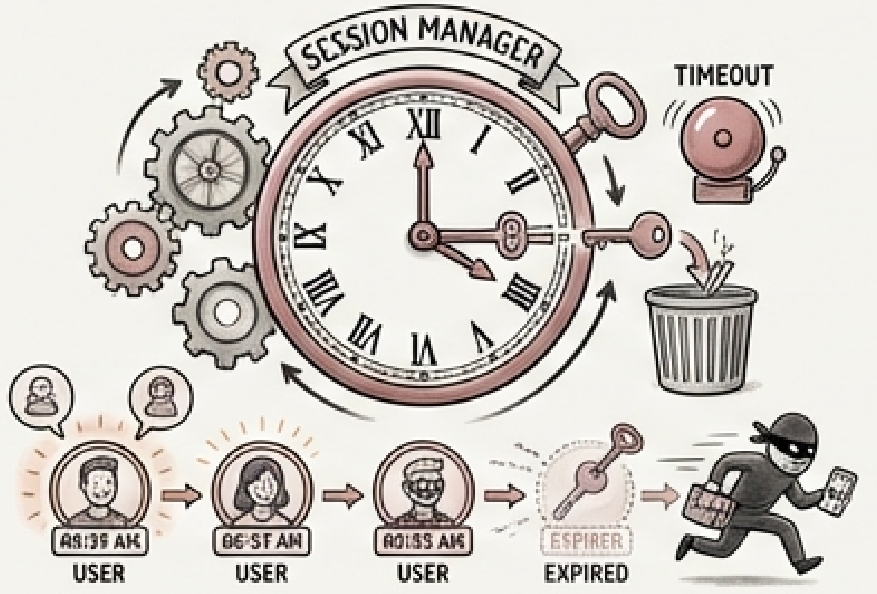


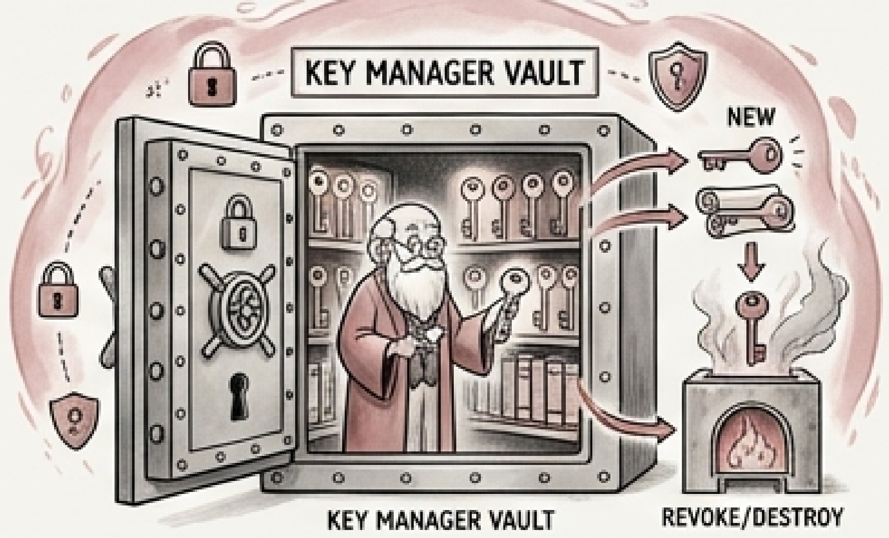
Secure Design Patterns: Building Blocks for Security





- **Autherization Enforcer:** Policy-based access control at service bou. enforcing least privilege.
- **Session Manager:** Secure session lifecycle management with timeout and rotation, preventing session hijacking.
- **Key Manager:** Centralized cryptographic key lifecycle management, protecting sensitive data.





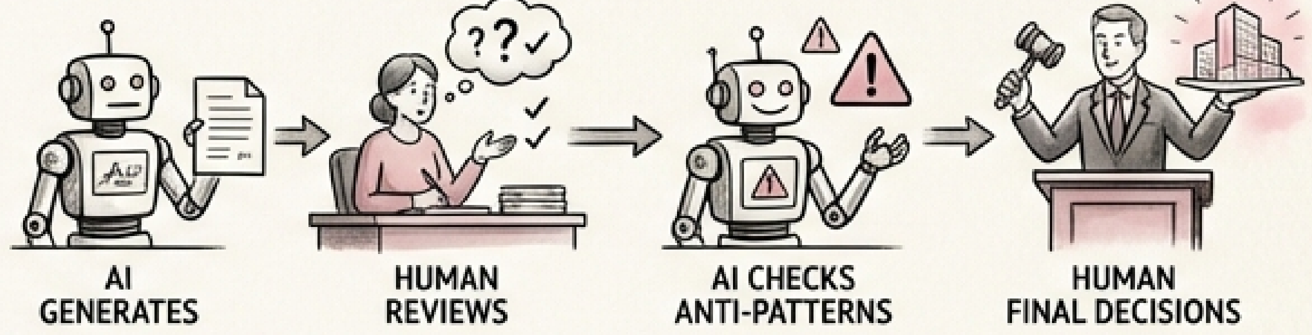
Anti-Patterns to Avoid: Common Security Pitfalls

- ⮾ Avoid implicit trust between services, requiring explicit authentication and authorization for all interactions.
- ⮾ Reject security through obscurity, relying on hidden code or configurations for protection.
- ⮾ Never enforce security solely on the client-side, as it can be easily bypassed.
- ⮾ Do not share credentials between users or services, as it increases the risk of compromise.
- ⮾ Avoid creating 'God objects' with excessive privileges, limiting the potential impact of a compromised component.

AI Architecture Review: Human Oversight is Crucial

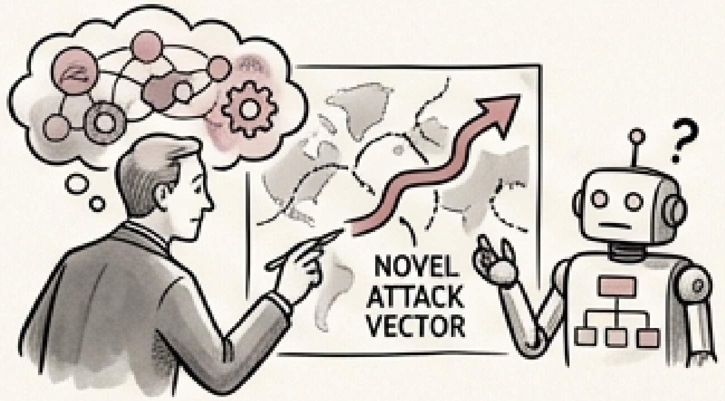
- AI tools can achieve 50-55% accuracy in detecting design flaws compared to experienced architects. Useful, but not sufficient alone.

- Best pattern: AI generates initial architecture -> human architect reviews -> AI checks for known anti-patterns -> human makes final decisions.



- AI tools are better at identifying known vulnerabilities and enforcing established security policies.

- Human architects are better at understanding complex interactions and identifying novel attack vectors.

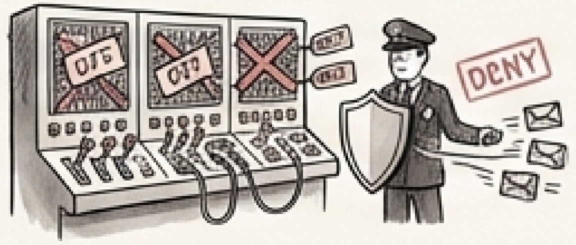


- Never let AI make irreversible architecture decisions autono- autonomously; human oversight is essential.

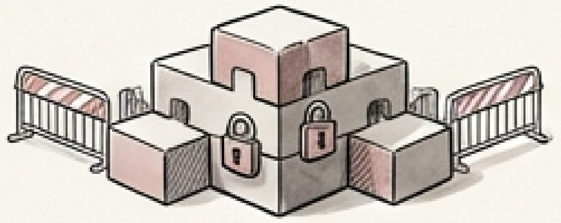
Key Takeaways: Securing AI-Augmented Systems

- Secure design principles are paramount: Design for **security** from the start.

- Conduct thorough **attack surface analysis** to identify potential vulnerabilities.

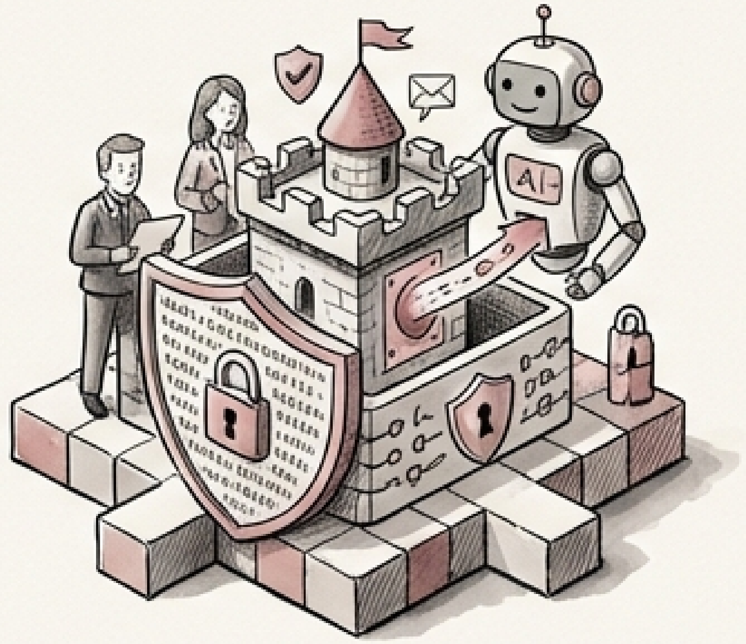


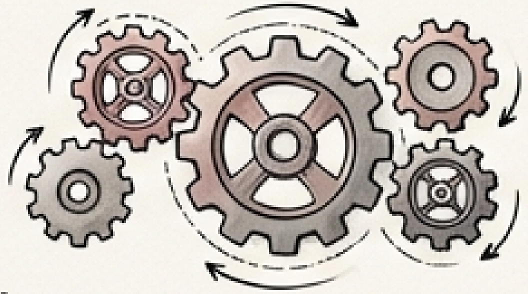
- Reduce the attack surface by **disabling unused features** and **enforcing strict input validation.**



- Use secure design patterns to build more **robust and secure systems.**

- Avoid common security anti-patterns, especially those introduced by **AI code generators.**





Q&A and Resources

- Open forum for questions and discussion.
- Links to OWASP AI Security and similar resources.
- Link to internal company security team or contact person.
- Suggested reading: NIST AI Risk Management Framework.
- Thank you for your time and attention.

THANK YOU

- Questions?