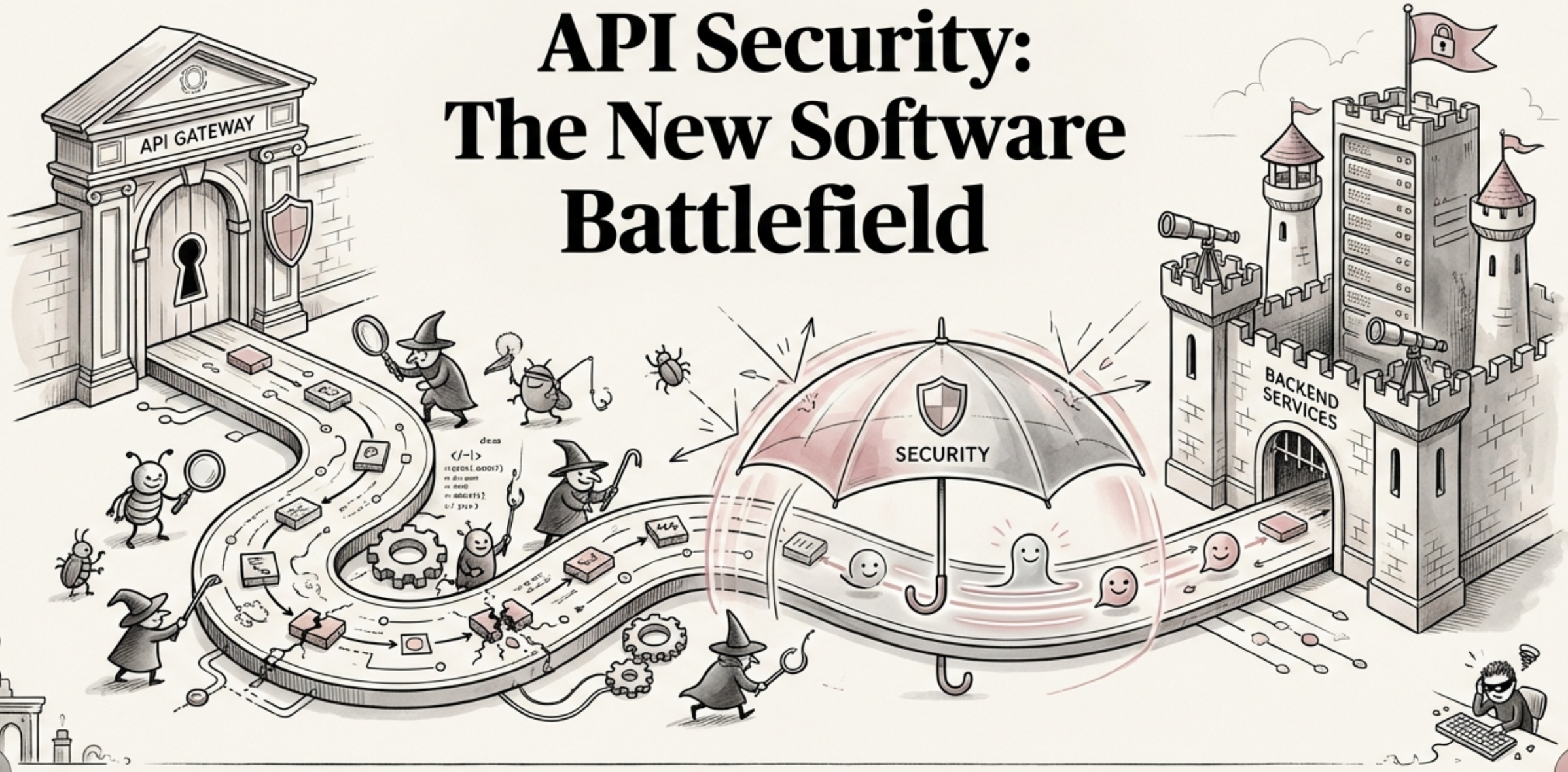
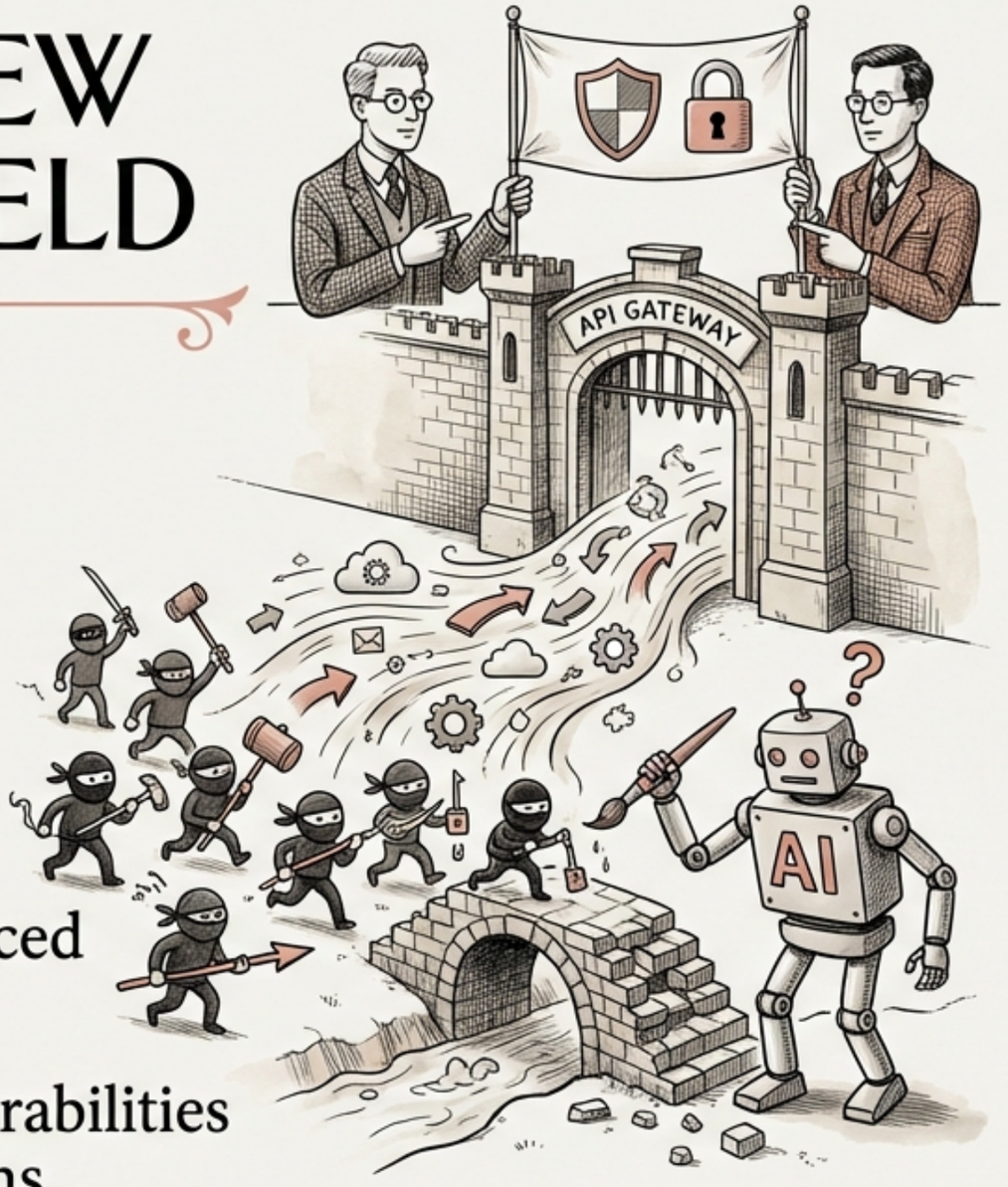
API Security: The New Software Battlefield

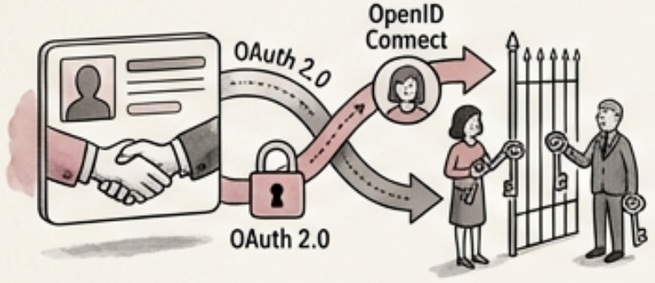


API SECURITY: THE NEW SOFTWARE BATTLEFIELD

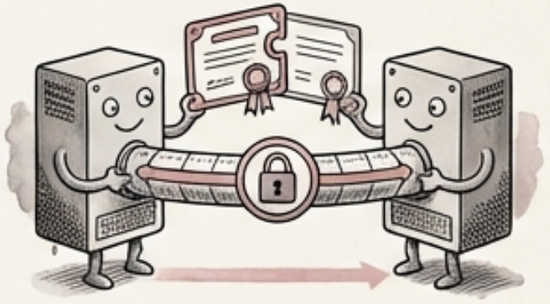
- APIs handle 83% of web traffic, making them a prime target for attacks.
- AI is accelerating API development, but often introduces insecure defaults.
- This module covers API security best practices from design to deployment.
- We will focus on mitigating risks introduced by AI-generated API code.
- Understanding and addressing API vulnerabilities is crucial for securing modern applications.



AUTHENTICATION AND AUTHORIZATION: SECURE API FOUNDATIONS

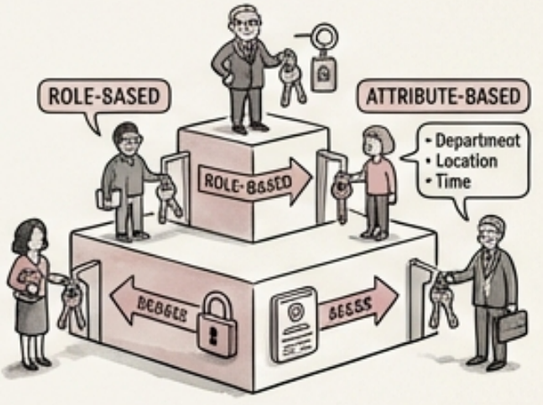


- OAuth 2.0 + OpenID Connect is recommended for user-facing API authentication.



- Mutual TLS (mTLS) should be used for secure service-to-service API communication.

- API keys are suitable for simple integrations but never as the sole authentication method.

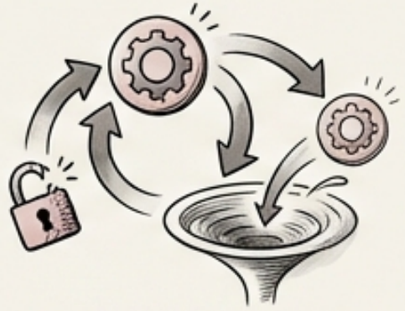


- Implement Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) for authorization.

- Enforce object-level permissions on every API request to prevent Broken Object Level Authorization (BOLA).

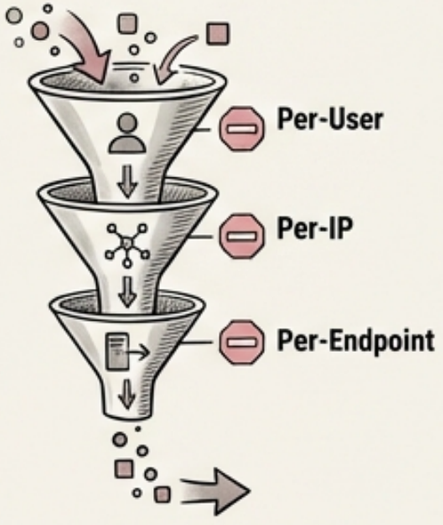
Token Management and Rate Limiting: Protecting API Resources

- Implement refresh token rotation to mitigate the impact of compromised refresh tokens.

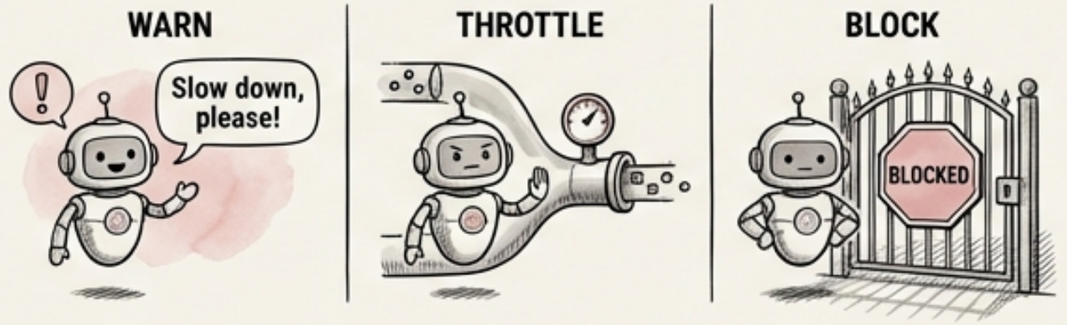


- Enable token revocation on logout to invalidate active sessions.

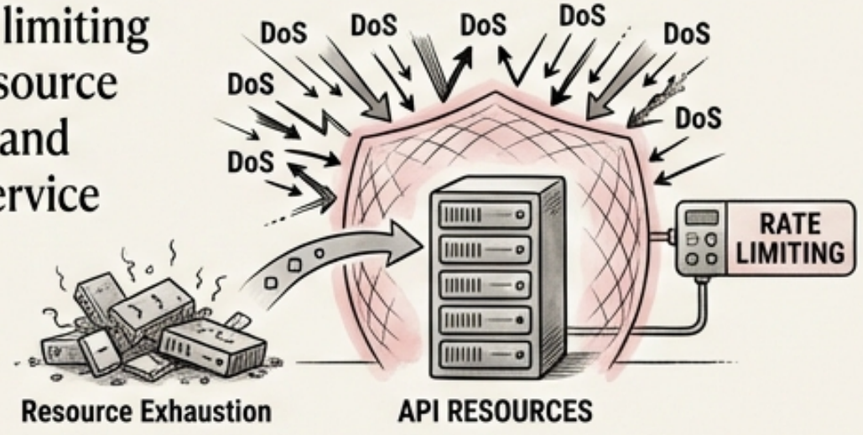
- Implement rate limiting on multiple levels: per-user, per-IP, and per-endpoint.



- Use a graduated response to rate limiting: warn, throttle, then block.

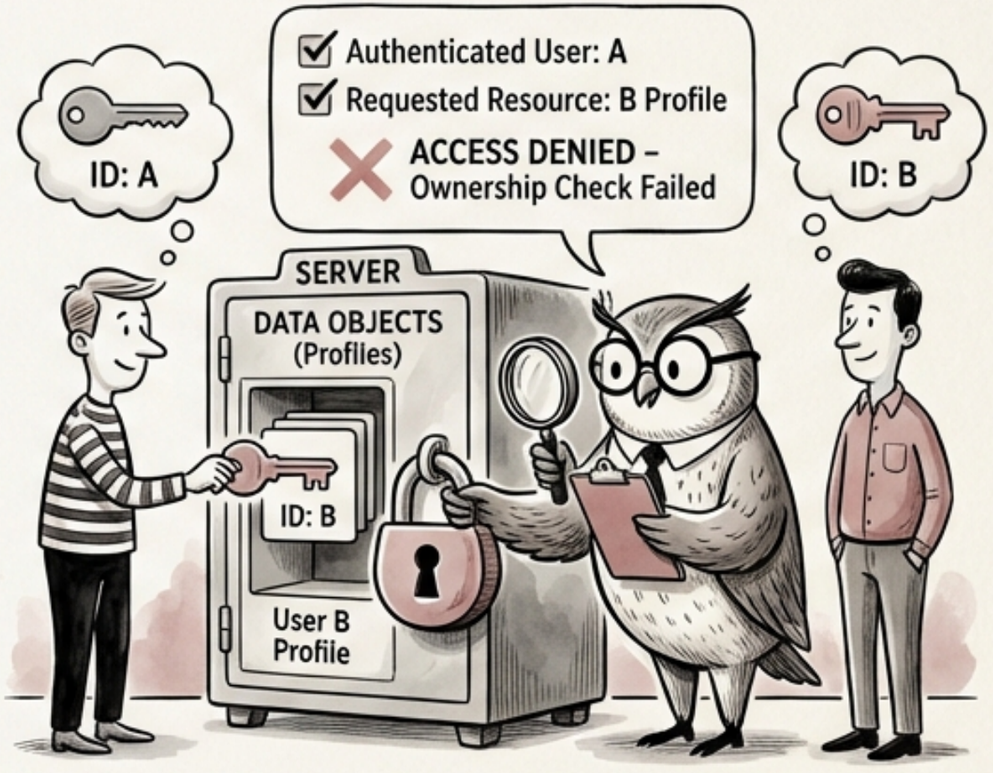


- Proper rate limiting prevents resource exhaustion and denial-of-service attacks.



OWASP API Security Top 10: Broken Object Level Authorization (API1)

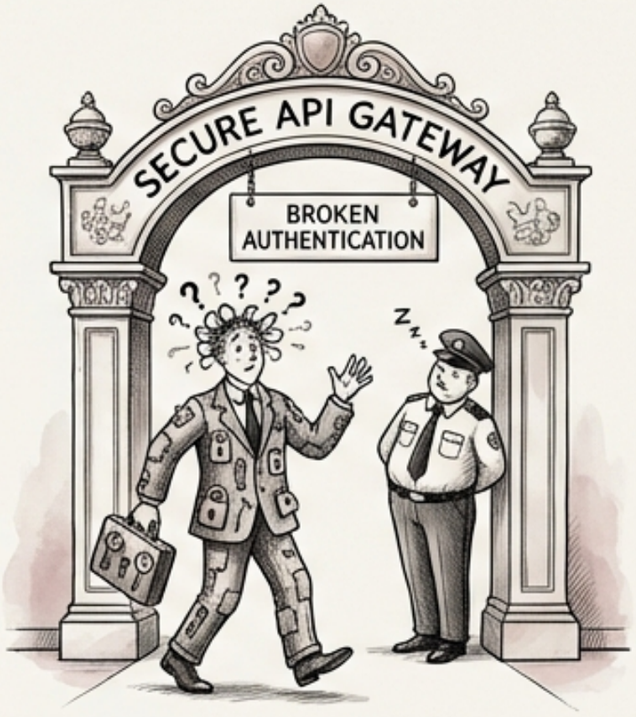
- API1 Broken Object Level Authorization (BOLA) is the most common API vulnerability.
- Always verify the requesting user owns the requested object on the server-side.
- Never trust client-supplied IDs without server-side ownership checks.
- Implement access control checks based on the authenticated user and the requested resource.
- Example: User A shouldn't be able to access User B's profile using User B's ID



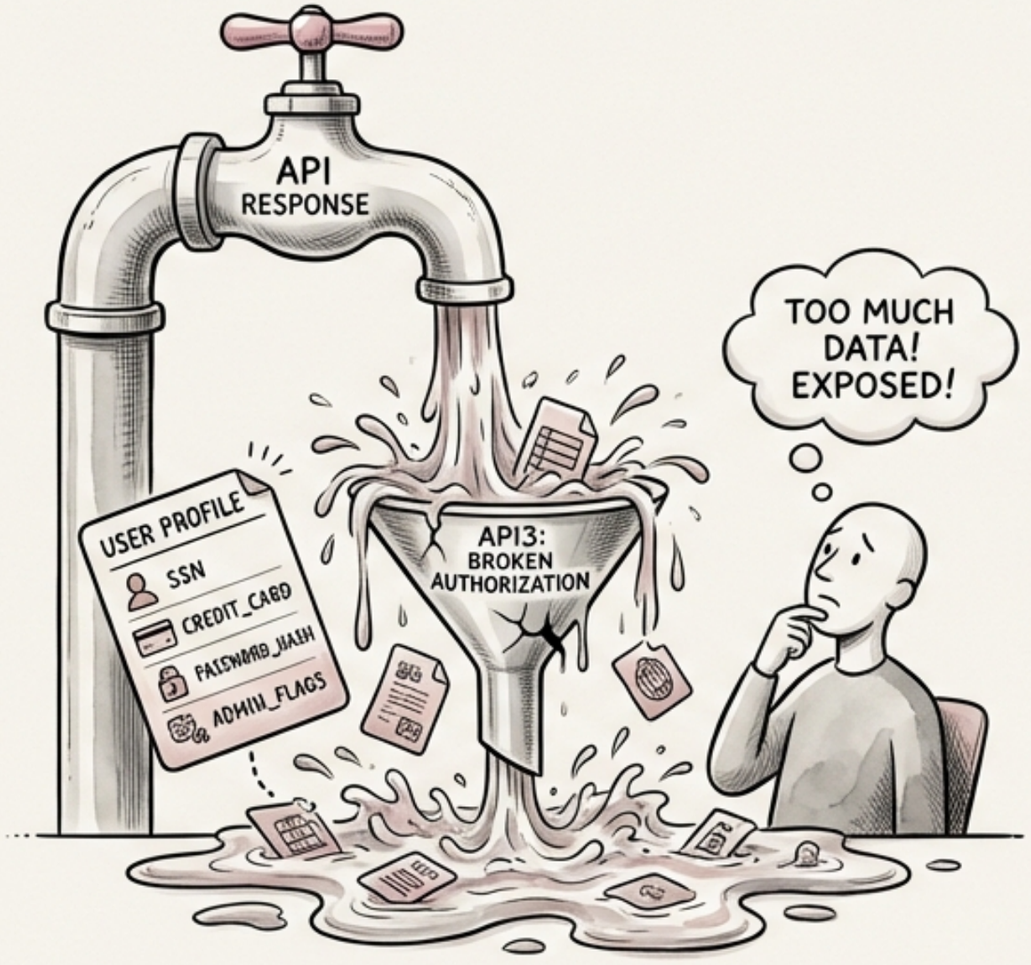
VISUAL METAPHOR: SERVER-SIDE OWNERSHIP VERIFICATION PREVENTS UNAUTHORIZED ACCESS.

OWASP API Security Top 10: Broken Authentication (API2)

- API2 Broken Authentication occurs when authentication mechanisms are flawed.
- Use proven authentication libraries to avoid common pitfalls.
- Implement proper JWT validation (signature, expiry, issuer, audience).
- Never roll your own custom authentication system.
- Ensure strong password policies and multi-factor authentication where appropriate.



OWASP API Security Top 10: Broken Object Property Level Authorization (API3)

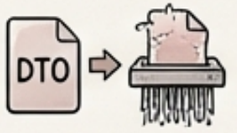


- API3 Broken Object Property Level Authorization exposes sensitive data through unfiltered API responses.

- Whitelist response fields explicitly to control which data is returned.

- Never return raw database objects directly to the client.

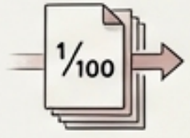
- Use Data Transfer Objects (DTOs) or serializers to shape the API response.



- Avoid over-fetching data from the database to minimize exposure of sensitive properties.

OWASP API Security Top 10: Unrestricted Resource Consumption (API4)

- API4 Unrestricted Resource Consumption leads to denial-of-service and resource exhaustion.



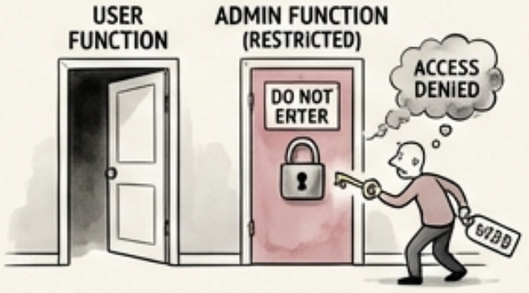
- Implement pagination with a maximum page size to limit the amount of data returned.

- Enforce file upload size limits to prevent large file uploads from consuming excessive resources.

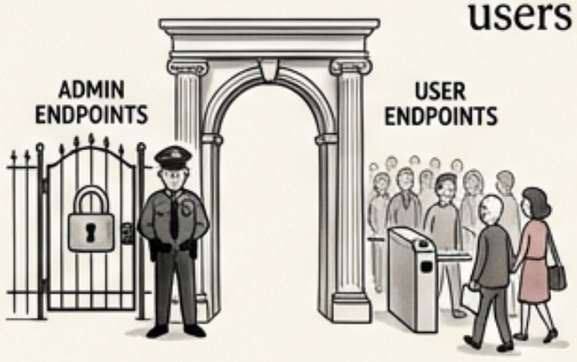
- Set query complexity limits to prevent complex queries from overloading the database.

- Enforce request timeouts to prevent long-running requests from tying up resources.

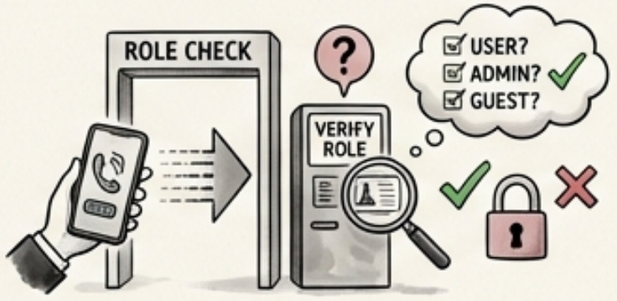
OWASP API Security Top 10: Broken Function Level Authorization (API5)



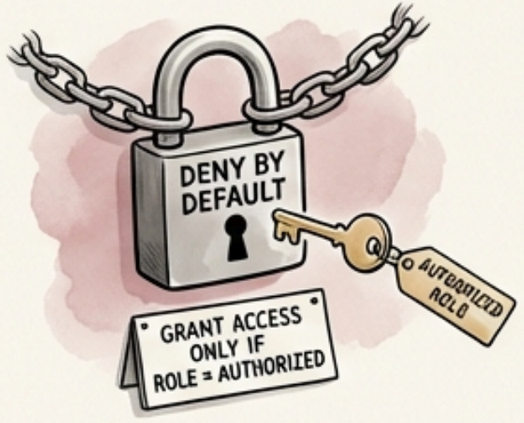
- API5 Broken Function Level Authorization occurs when users can access functions they aren't authorized to use.



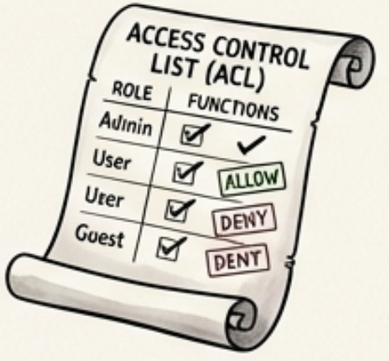
- Separate administrative and user endpoints to isolate privileged functions.



- Verify the user's role on every function call to ensure they have the necessary permissions.



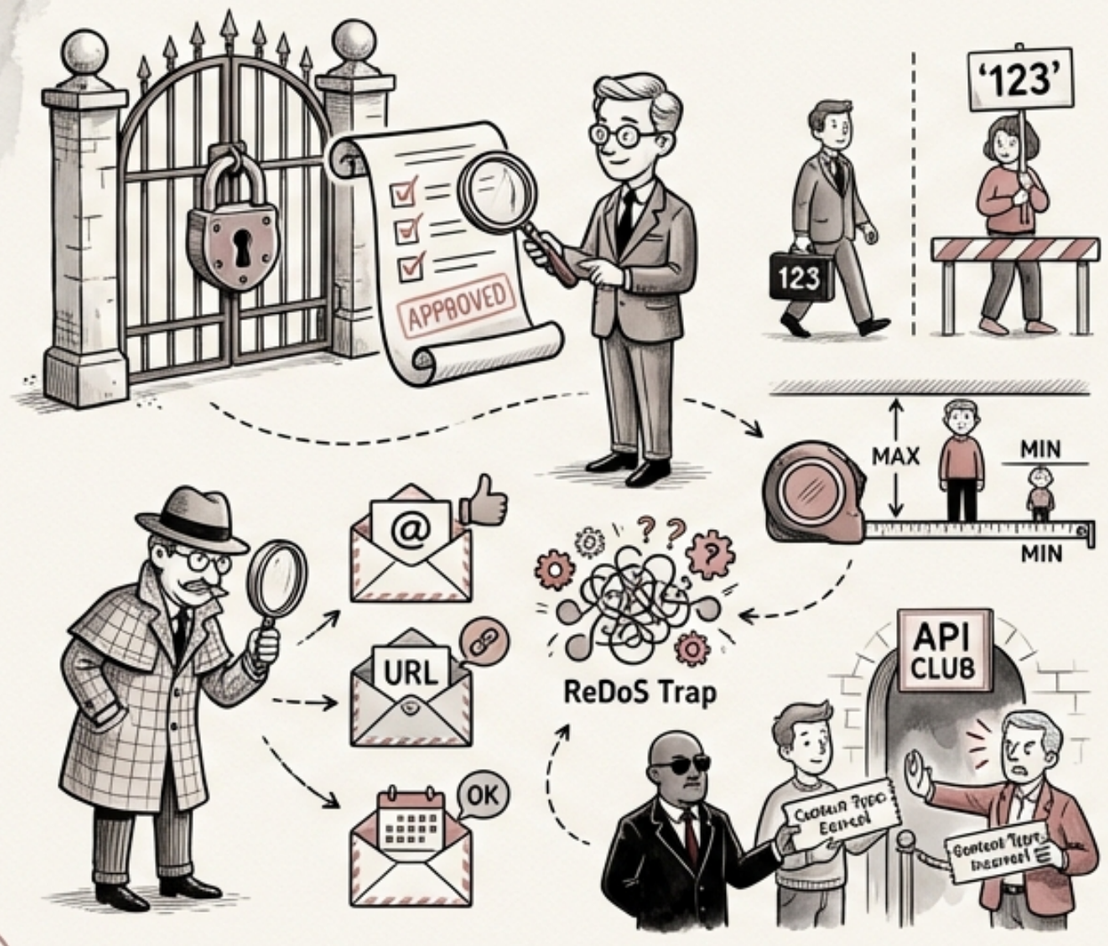
- Implement a deny-by-default policy, only granting access to specific functions based on role.



- Use a well-defined access control list (ACL) to manage function-level permissions.

Input Validation: A Multi-Layered API Defense

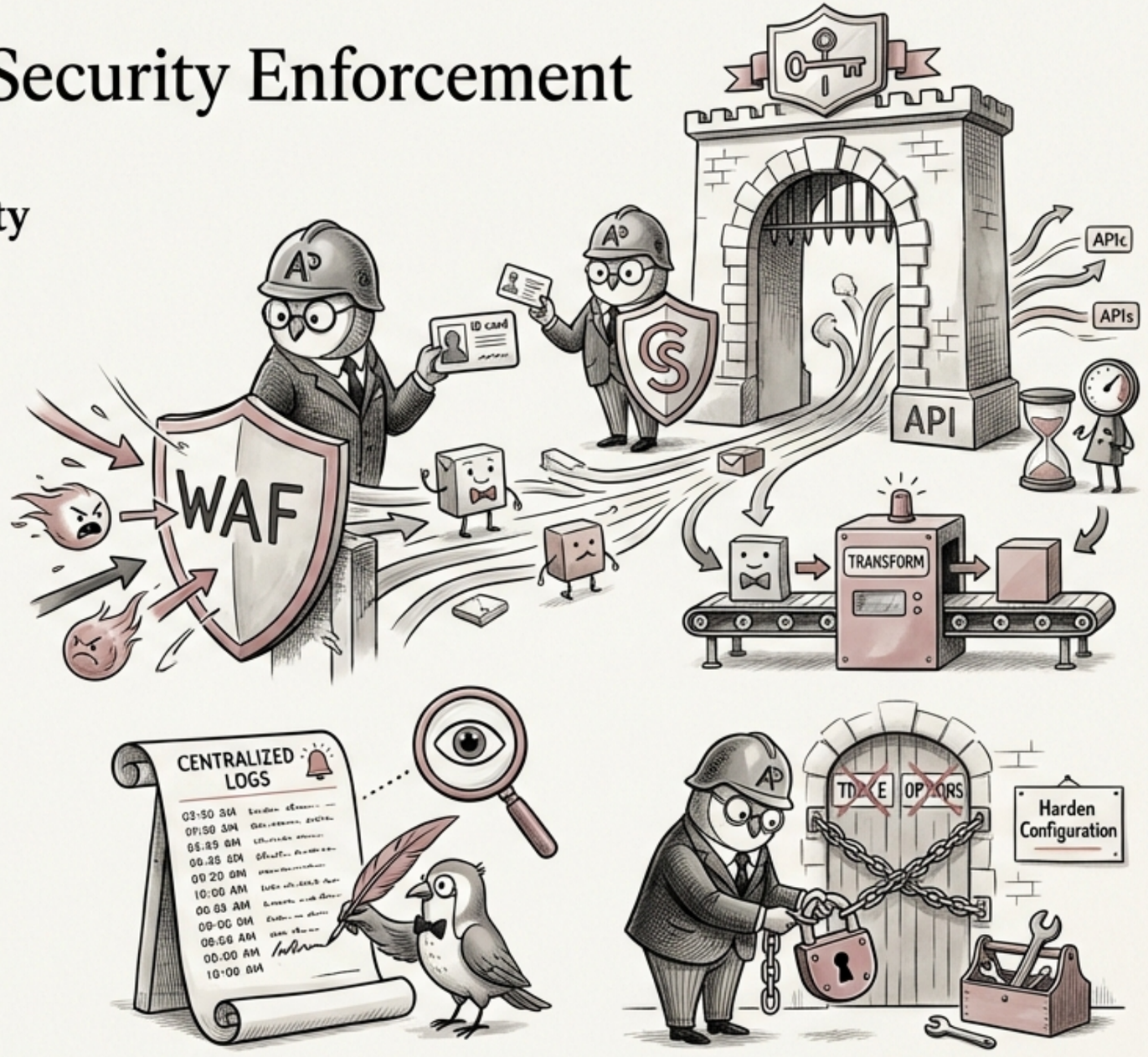
Securing Your Digital Fortress



- Enforce **OpenAPI/JSON Schema validation** on every request to reject invalid data.
- Use **strict type checking** with no implicit conversion (e.g., string '123' is not integer 123).
- Implement **min/max range validation** for all **numeric fields** and **length limits** for strings.
- Use **regex patterns** for format validation (emails, URLs, dates), but be cautious of **ReDoS attacks**.
- **Reject** requests with incorrect **Content-Type** headers.

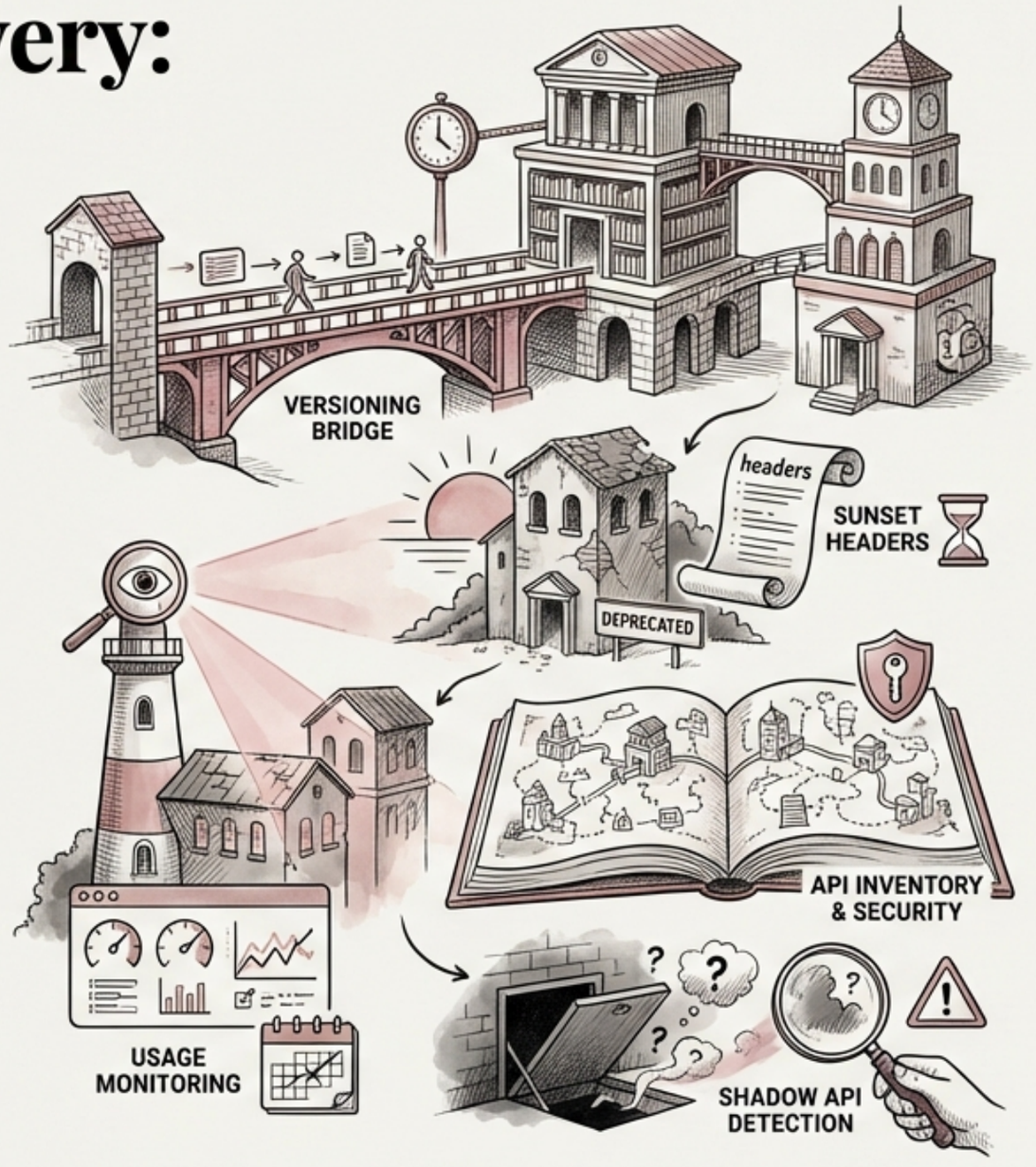
API Gateway: Centralized Security Enforcement

- API Gateways provide **centralized security enforcement** for APIs.
- They handle **authentication verification**, rate limiting, and request/response **transformation**.
- Integrate a **Web Application Firewall** (WAF) with the API gateway for additional protection.
- Enable centralized logging for security monitoring and incident response.
- Harden the gateway configuration by disabling unnecessary HTTP methods (TRACE, OPTIONS).



API Versioning and Discovery: Managing API Lifecycle

- Maintain backward compatibility during API changes through versioning.
- Deprecate old API versions with sunset headers to signal end-of-life.
- Monitor usage of deprecated endpoints to plan for removal.
- Maintain an accurate API inventory for visibility and security assessment.
- Detect shadow APIs (APIs deployed without proper oversight) to mitigate risks.



AI-GENERATED API SECURITY CHECKLIST

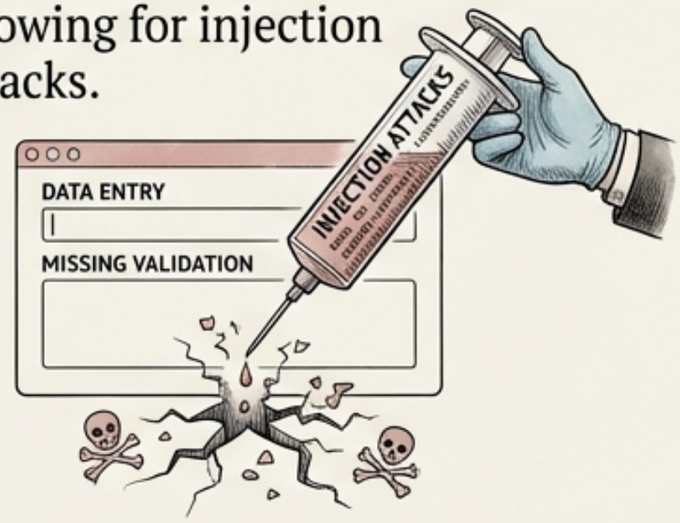
1. AI-generated APIs often lack authentication, leading to unauthorized access.

2. Overly permissive CORS configurations (Access-Control-Allow-Origin: *) are common.

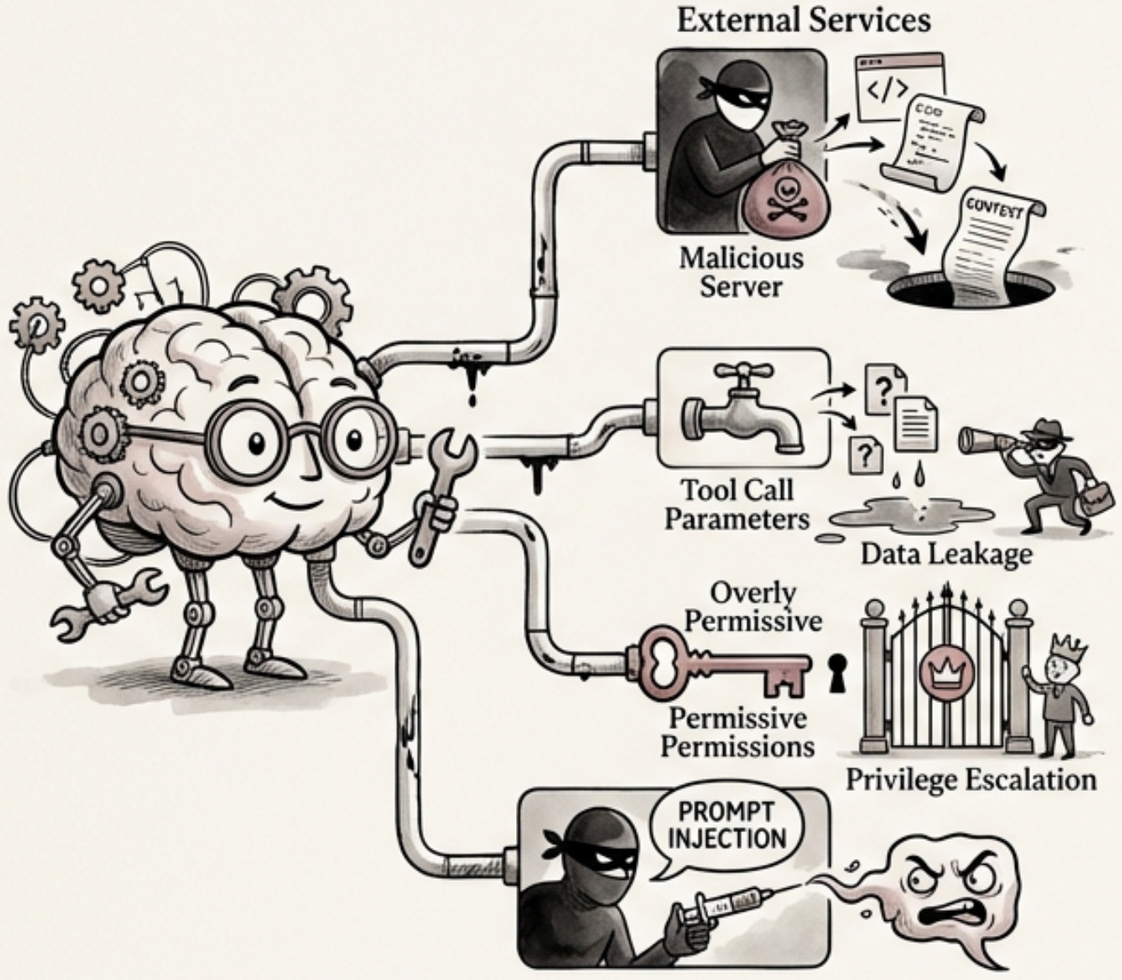
3. Rate limiting is frequently absent, making APIs vulnerable to abuse.

4. Verbose error

5. Input validation is often missing, allowing for injection attacks.



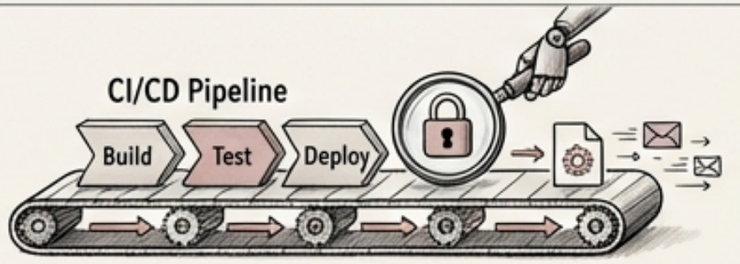
MCP (Model Context Protocol) Security: Securing AI Tool Interactions

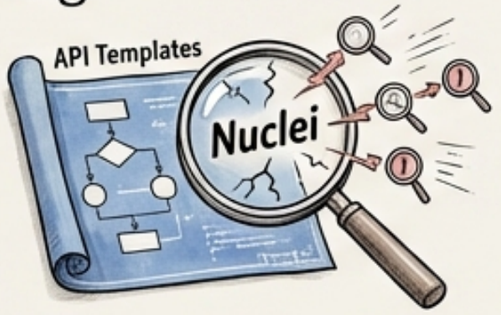


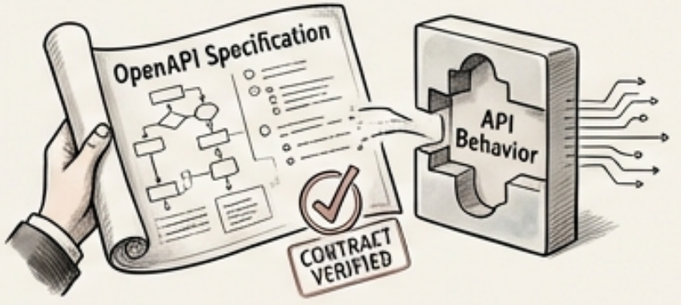
- Model Context Protocol (MCP) enables AI tools to connect to external services.
- Malicious MCP servers can exfiltrate code and context, posing a significant risk.
- Data leakage can occur through tool call parameters if not properly sanitized.
- Overly permissive tool permissions can lead to privilege escalation.
- **Prompt injection** through tool response manipulation is a new attack vector.

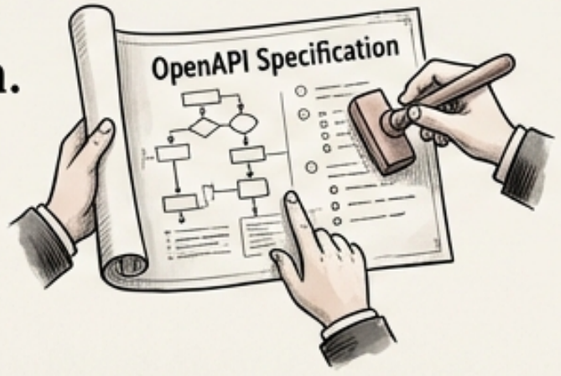
Automated API Security Testing in CI/CD Pipelines

- ✓ Integrate API security testing into the CI/CD pipeline for continuous security.
- ✓ Use Dynamic Application Security Testing (DAST) tools like OWASP ZAP and Burp Suite for API scanning.
- ✓ Utilize Nuclei with API templates for automated vulnerability detection.
- ✓ Implement contract testing to verify API behavior matches the OpenAPI specification.
- ✓ Use fuzz testing to send malformed inputs and injection payloads to every endpoint.



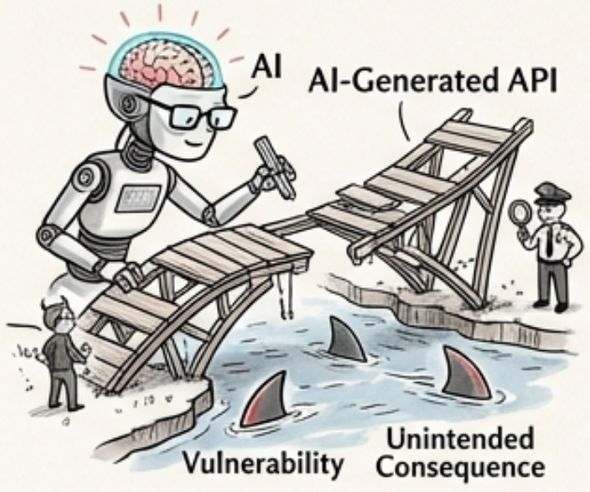


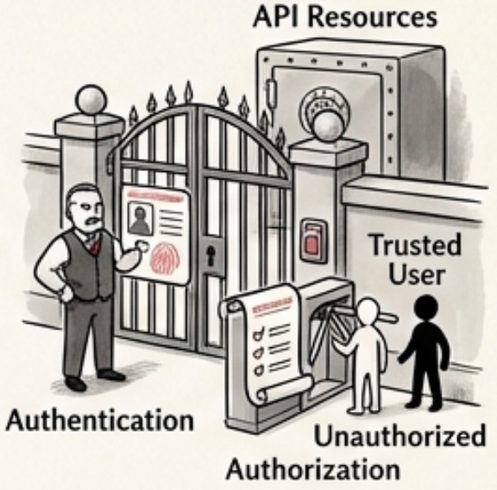




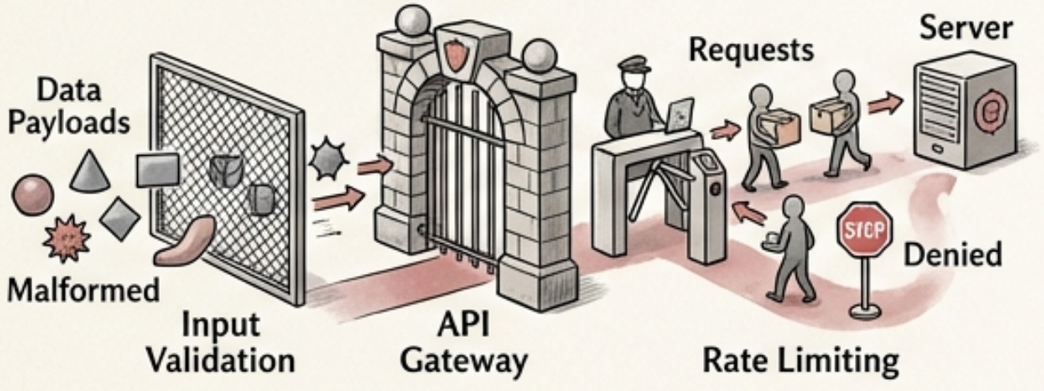
Securing APIs: A Continuous Journey

- **API security** is a critical aspect of modern application development.
- **AI-generated APIs** introduce new security challenges that require careful attention.





- Implement robust **authentication** and authorization mechanisms to protect API resources.
- Utilize **input validation, API gateways,** and **rate limiting** for defense in depth.



- Regularly **review** and **update** API security configurations to address evolving threats.

Thank You

- Questions?