Web App

Owner: Michael Amponsem
Reviewer: Thomas Darko
Contributors:
Date Generated: Wed Aug 21 2024

Executive Summary

High level system description

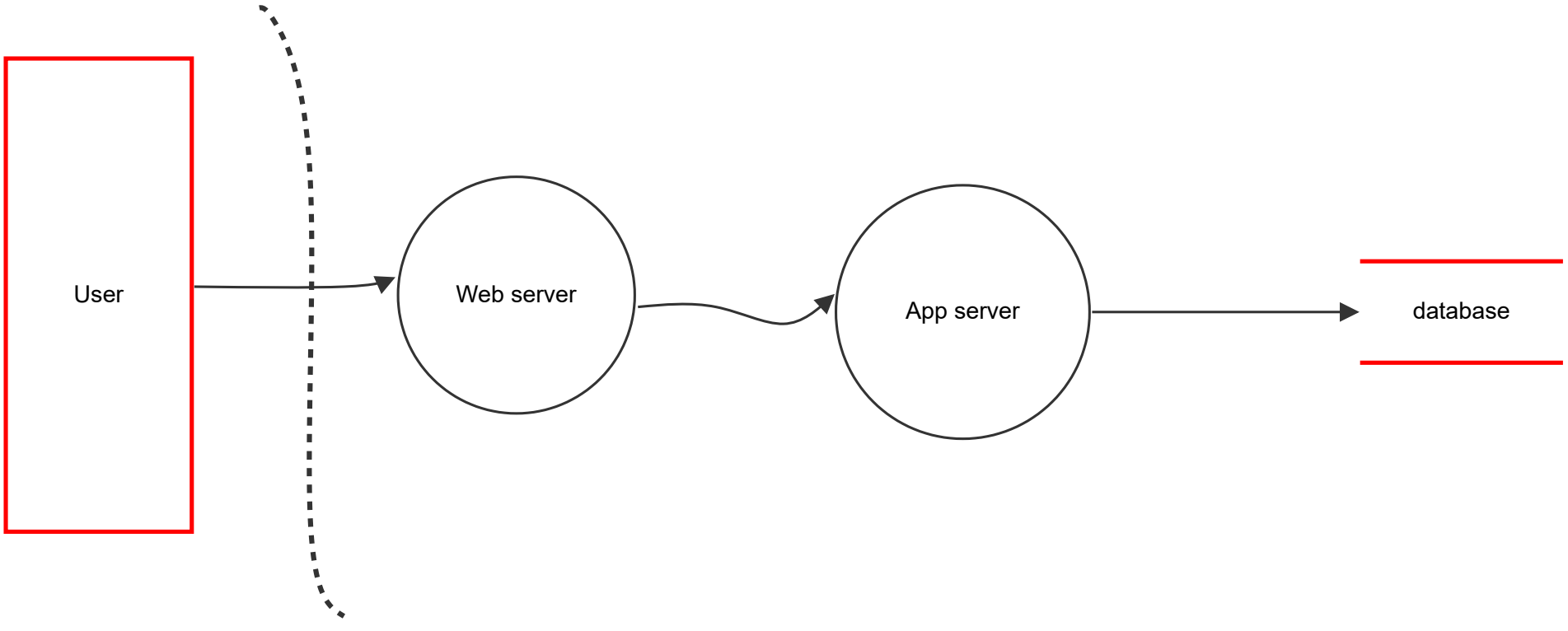
Web Security Fundamentals Lab

Summary

Total Threats	15
Total Mitigated	12
Not Mitigated	3
Open / High Priority	1
Open / Medium Priority	2
Open / Low Priority	0
Open / Unknown Priority	0

WebApp

Conduct a threat modelling exercise for a web application



WebApp

User (Actor)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
17	New STRIDE threat	Spoofing	High	Open	9/10	An attacker could impersonate another user or a trusted entity, leading to unauthorized access or deception.	Implement multi-factor authentication (MFA) and encourage the use of strong, unique passwords
18	New STRIDE threat	Repudiation	Medium	Mitigated	8/10	Users could deny performing actions within the system, leading to disputes.	Encourage users to keep their devices and software up to date to minimize vulnerabilities.

Data Flow (Data Flow)

Number	Title	Type	Priority	Status	Score	Description	Mitigations

Data Flow (Data Flow)

Number	Title	Type	Priority	Status	Score	Description	Mitigations

Data Flow (Data Flow)

Number	Title	Type	Priority	Status	Score	Description	Mitigations

Web server (Process)

The web server ensures that users can access web content and applications reliably and securely

Number	Title	Type	Priority	Status	Score	Description	Mitigations
3	New STRIDE threat	Spoofing	High	Mitigated	9/10	Faking to be someone else, usually to gain unauthorized access	Implement strong authentication mechanisms and multi-factor authentication (MFA).
4	New STRIDE threat	Information disclosure	High	Mitigated	8/10	Unauthorized access to information	Protecting sensitive information from unauthorized access is essential for data privacy.
5	New STRIDE threat	Tampering	High	Mitigated	9/10	Unauthorized changes to data or systems.	secure coding practices and encryption.
6	New STRIDE threat	Denial of service	Medium	Mitigated	7/10	Disrupting the availability of a service	Implement load balancing

App server (Process)

Receives and process request from a user.

Number	Title	Type	Priority	Status	Score	Description	Mitigations
12	New STRIDE threat	Spoofing	Medium	Mitigated	9/10	An attacker could impersonate a legitimate user to gain unauthorized access to the application server.	Implement strong authentication mechanisms, such as multi-factor authentication (MFA) and secure API keys
13	New STRIDE threat	Tampering	Medium	Mitigated	8/10	An attacker might attempt to modify the application server's files, configurations, or data in transit.	Apply strong access controls.
14	New STRIDE threat	Repudiation	Medium	Mitigated	8/10	Users or services could deny performing actions on the application server	Implement comprehensive logging and auditing of all actions on the application server
15	New STRIDE threat	Information disclosure	Medium	Mitigated	8/10	Sensitive data processed or stored by the application server could be exposed to unauthorized parties, either through vulnerabilities in the application code or through insecure configurations.	Use secure coding practices to prevent vulnerabilities such as SQL injection or cross-site scripting (XSS).
16	New STRIDE threat	Denial of service	Medium	Mitigated	7/10	An attacker might try to overwhelm the application server with requests, making it unavailable to legitimate users or severely degrading its performance.	Implement load balancing, and redundancy to handle large volumes of traffic.

database (Store)

Stores users credentials.

Number	Title	Type	Priority	Status	Score	Description	Mitigations
8	New STRIDE threat	Tampering	High	Mitigated	9/10	An attacker could attempt to modify data in the database	Enforce access controls and permissions to prevent unauthorized modifications.
9	New STRIDE threat	Repudiation	Medium	Mitigated	7/10	A user could perform actions in the database and later deny having done so	Ensure comprehensive logging of all database activities
10	New STRIDE threat	Information disclosure	Medium	Open	8/10	Sensitive data in the database could be exposed to unauthorized parties, leading to data breaches.	Enforce strict access controls, and use data masking techniques to limit the exposure of sensitive information.
11	New STRIDE threat	Denial of service	Medium	Open	7/10	An attacker might attempt to overload the database, making it unavailable to legitimate users or severely degrading its performance.	enforce strict access controls, and use data masking techniques to limit the exposure of sensitive information.

Here's a breakdown of how each OWASP Top 10 vulnerability could impact The RESTful APIs, along with practical examples and mitigations:

1. Broken Access Control

Vulnerability: APIs might not enforce proper access controls, allowing unauthorized users to access or modify data.

Example: An API endpoint like /admin/users that lists all users might be accessible by any authenticated user, not just administrators.

Mitigation: Implement robust access control mechanisms. Use role-based access control (RBAC) and ensure endpoints are protected by checking user permissions before performing sensitive operations.

2. Cryptographic Failures

Vulnerability: APIs might not properly secure sensitive data in transit or at rest, leading to exposure of sensitive information.

Example: An API that sends passwords or personal data over HTTP instead of HTTPS, exposing data to potential interception.

Mitigation: Use strong encryption protocols (e.g., TLS) for data in transit and ensure sensitive data is encrypted at rest. Regularly review cryptographic practices to align with current standards.

3. Injection

Vulnerability: APIs may be vulnerable to injection attacks, such as SQL injection or command injection, if user inputs are not properly sanitized.

Example: An API endpoint that allows users to search by name using a parameter directly included in a SQL query without proper escaping, such as /search?name=ampons.

Mitigation: Use parameterized queries or prepared statements to interact with databases. Validate and sanitize all user inputs to prevent injection attacks.

4. Insecure Design

Vulnerability: The API design might inherently have security flaws due to improper design choices or lack of security considerations.

Example: An API that exposes detailed error messages or stack traces in production, which could provide attackers with useful information for exploitation.

Mitigation: Follow secure design principles and conduct threat modeling during the design phase. Avoid exposing sensitive details in error messages and follow the principle of least privilege.

5. Security Misconfiguration

Vulnerability: API configurations might be improperly set, leading to security weaknesses.
Example: Default security configurations that allow unauthenticated access to endpoints or exposed management interfaces.
Mitigation: Review and harden configurations, disable unnecessary features, and use security best practices. Regularly update and patch your API components.

6. Vulnerable and Outdated Components

Vulnerability: APIs might use outdated or vulnerable libraries and components that have known security issues.

Example: An API using an outdated version of a third-party library with known vulnerabilities.

Mitigation: Regularly update libraries and dependencies to their latest secure versions. Use tools like dependency checkers to identify and address vulnerabilities in components.

7. Identification and Authentication Failures

Vulnerability: APIs might have weak or improperly implemented authentication mechanisms, leading to unauthorized access.

Example: An API that allows login with weak passwords or doesn't enforce multi-factor authentication (MFA).

Mitigation: Implement strong authentication mechanisms, enforce strong password policies, and use MFA where appropriate. Ensure authentication tokens are securely managed.

8. Software and Data Integrity Failures

Vulnerability: APIs might lack integrity checks for software updates or data, leading to potential tampering.

Example: An API that accepts file uploads without validating the integrity of the files.

Mitigation: Implement integrity checks for software updates and data uploads. Use checksums, signatures, or other mechanisms to ensure data integrity.

9. Security Logging and Monitoring Failures

Vulnerability: APIs might not log security events or monitor for suspicious activities effectively.

Example: An API that does not log failed login attempts, making it difficult to detect brute force attacks.

Mitigation: Implement comprehensive logging and monitoring for security events. Ensure logs are protected and reviewed regularly to detect and respond to potential security incidents.

10. Server-Side Request Forgery (SSRF)

Vulnerability: APIs might allow attackers to make unauthorized requests to internal systems or services.

Example: An API endpoint that allows users to specify a URL to fetch data without validating the URL, potentially allowing access to internal services.

Mitigation: Validate and sanitize user inputs, especially when they are used to make requests to other services. Implement proper access controls and network segmentation to mitigate SSRF risks.

Summary

To enhance RESTful API security, it's crucial to address these vulnerabilities by following best practices and leveraging security tools and frameworks. Regularly audit and update security measures to stay protected against emerging threats.

Semgrep Scanned Image