RESTful API with basic web security

Owner: Dennis Owusu
Reviewer: Thomas Darko
Contributors:
Date Generated: Wed Aug 21 2024

Executive Summary

High level system description

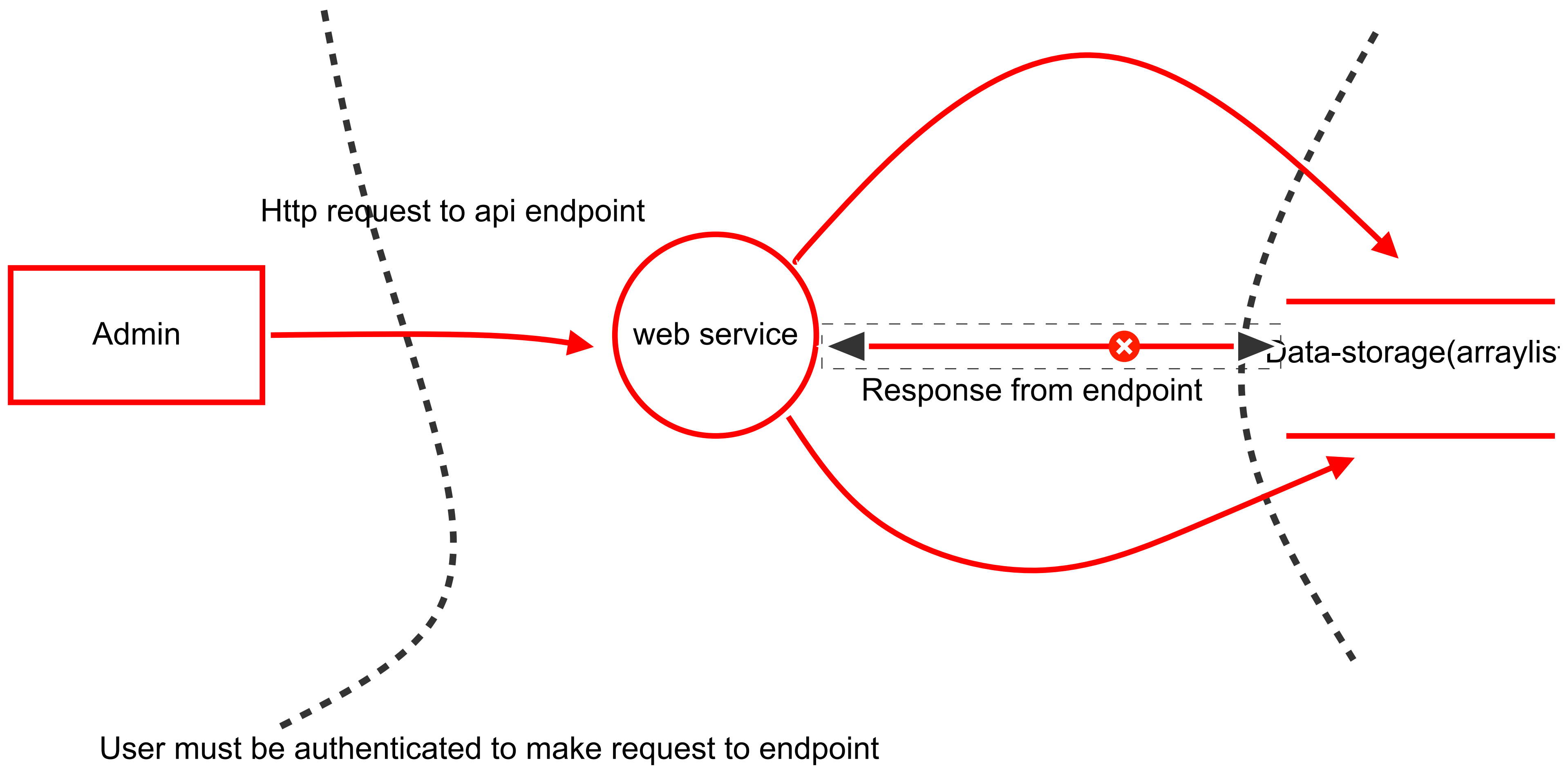
A RESTful API built using spring boot which implements basic security measures.

Summary

Total Threats	13
Total Mitigated	0
Not Mitigated	13
Open / High Priority	12
Open / Medium Priority	1
Open / Low Priority	0
Open / Unknown Priority	0

RESTful api diagram

The diagram shows how various components of the api interact with oneanother



Http request to api endpoint
Admin
web service
Data-storage(arraylis
Response from endpoint
User must be authenticated to make request to endpoint

RESTful api diagram

Admin (Actor)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
0	New STRIDE threat	Spoofing	Medium	Open		Provide a description for this threat	Provide remediation for this threat or a reason if status is N/A

web service (Process)

Desktop client for making http requests

Number	Title	Type	Priority	Status	Score	Description	Mitigations
4	New STRIDE threat	Spoofing	High	Open		An attacker might impersonate a legitimate user using stolen credentials	o Use strong authentication mechanisms like JWT tokens or OAuth2. o Implement multi-factor authentication (MFA). o Monitor and audit login attempts for suspicious activity.
5	New STRIDE threat	Denial of service	High	Open		An attacker could flood the API with requests, making it unavailable to legitimate users.	o Deploy a Web Application Firewall (WAF) to filter malicious traffic

Data-storage(arraylist) (Store)

An arraylist that stores a list of created users

Number	Title	Type	Priority	Status	Score	Description	Mitigations
6	New STRIDE threat	Tampering	High	Open		An attacker could modify data in transit between the client and server	o Enforce HTTPS for all communications to encrypt data in transit. o Use HMAC (Hash-based Message Authentication Code) to ensure data integrity.
7	New STRIDE threat	Information disclosure	High	Open		Sensitive information could be inadvertently exposed to unauthorized users	o Apply strict access controls using role-based access control (RBAC). o Sanitize and validate all inputs to prevent injection attacks. o Encrypt sensitive data both at rest and in transit.

Data Flow (Data Flow)

Post request that creates a user and stores the user in an arraylist

Number	Title	Type	Priority	Status	Score	Description	Mitigations
12	New STRIDE threat	Tampering	High	Open		An attacker could modify data in transit between the client and server.	o Enforce HTTPS for all communications to encrypt data in transit. o Use HMAC (Hash-based Message Authentication Code) to ensure data integrity.

Number	Title	Type	Priority	Status	Score	Description	Mitigations
13	New STRIDE threat	Information disclosure	High	Open		Sensitive information could be inadvertently exposed to unauthorized users	o Apply strict access controls using role-based access control (RBAC). o Sanitize and validate all inputs to prevent injection attacks. o Encrypt sensitive data both at rest and in transit.
14	New STRIDE threat	Denial of service	High	Open		An attacker could flood the API with requests, making it unavailable to legitimate users.	o Implement rate limiting and IP blocking. o Deploy a Web Application Firewall (WAF) to filter malicious traffic. o Ensure scalable infrastructure to handle potential traffic surges.

Data Flow (Data Flow)

An admin can make post and get request to the api

Number	Title	Type	Priority	Status	Score	Description	Mitigations
15	New STRIDE threat	Tampering	High	Open		An attacker could modify data in transit between the client and server.	o Enforce HTTPS for all communications to encrypt data in transit. o Use HMAC (Hash-based Message Authentication Code) to ensure data integrity.
16	New STRIDE threat	Denial of service	High	Open		An attacker could flood the API with requests, making it unavailable to legitimate users.	o Implement rate limiting and IP blocking. o Deploy a Web Application Firewall (WAF) to filter malicious traffic. o Ensure scalable infrastructure to handle potential traffic surges.

Data Flow (Data Flow)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
19	New STRIDE threat	Denial of service	High	Open		An attacker could flood the API with requests, making it unavailable to legitimate users.	o Implement rate limiting and IP blocking. o Deploy a Web Application Firewall (WAF) to filter malicious traffic. o Ensure scalable infrastructure to handle potential traffic surges.

Data Flow (Data Flow)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
17	New STRIDE threat	Information disclosure	High	Open		Sensitive information could be inadvertently exposed to unauthorized users.	o Apply strict access controls using role-based access control (RBAC). o Sanitize and validate all inputs to prevent injection attacks. o Encrypt sensitive data both at rest and in transit.
18	New STRIDE threat	Tampering	High	Open		An attacker could modify data in transit between the client and server.	o Enforce HTTPS for all communications to encrypt data in transit. o Use HMAC (Hash-based Message Authentication Code) to ensure data integrity.

In my Spring Boot project, several basic security measures were applied to ensure the API was secure. Here's a summary of those security measures:

1. Authentication

- **Basic Authentication: implemented basic authentication using httpBasic(). This ensures that only authenticated users can access the API endpoints.**
- **UserDetailsService: A custom UserDetailsService was provided to load user-specific data.Used InMemoryUserDetailsManager for storing user details in memory, which is suitable for testing or small-scale applications.**

2. Authorization

- **Role-Based Access Control (RBAC): The API was configured to ensure that only authenticated users can access any endpoint. This was done using authorizeHttpRequests().anyRequest().authenticated().**

3. CSRF Protection

- **CSRF (Cross-Site Request Forgery) Protection Disabled: Disabled CSRF protection using csrf(AbstractHttpConfigurer::disable) for simplicity. In production, CSRF protection would be enabled, especially for state-changing requests like POST, PUT, and DELETE.**

4. Password Encoding

- **Password Storage with No Encoding for Testing: For demonstration purposes, I stored the password using {noop} prefix to indicate no encoding. In a production environment, passwords should be stored securely using a strong encoder like BCryptPasswordEncoder.**

5. Input Validation

- **Input Validation: implemented basic input validation to ensure that incoming data conforms to expected formats, reducing the risk of injection attacks.**

6. Output Encoding

- **Output Encoding: Ensured that data returned by the API is properly encoded to prevent cross-site scripting (XSS) attacks. This measure ensures that any user-generated content is safe to display in web pages.**

7. Security Configuration

- **SecurityFilterChain: used SecurityFilterChain to customize the security settings, including disabling CSRF and configuring basic authentication.**

Web Security Fundamentals: A Focus on the OWASP Top 10

1. Introduction to Web Security

Web security is crucial for protecting web applications from threats and vulnerabilities that could lead to data breaches, unauthorized access, and other cyber threats. The OWASP (Open Web Application Security Project) Top 10 is a standard awareness document that highlights the most critical security risks to web applications.

2. Overview of the OWASP Top 10

The OWASP Top 10 is a list of the most critical security risks facing web applications. It serves as a foundation for web security best practices and helps developers, security professionals, and organizations prioritize their security efforts.

3. The OWASP Top 10 Vulnerabilities

A1: Broken Access Control

- **Description:** Occurs when restrictions on what authenticated users are allowed to do are not properly enforced.
- **Example:** A user accessing another user's account by modifying the URL or user ID.
- **Mitigation:** Implement role-based access control (RBAC) and validate user permissions on the server side.

A2: Cryptographic Failures

- **Description:** Inadequate encryption or improper handling of sensitive data.
- **Example:** Using weak encryption algorithms or failing to encrypt sensitive data.
- **Mitigation:** Use strong, up-to-date cryptographic algorithms and ensure all sensitive data is encrypted.

A3: Injection

- **Description:** Occurs when untrusted data is sent to an interpreter as part of a command or query.
- **Example:** SQL Injection, where an attacker can manipulate SQL queries to execute arbitrary commands.
- **Mitigation:** Use parameterized queries, prepared statements, and input validation.

A4: Insecure Design

- **Description:** Flaws in the design of an application can lead to vulnerabilities, even if the code itself is secure.
- **Example:** Designing a system without considering security risks.

- **Mitigation:** Integrate security into the design process, threat model your application, and follow secure design principles.

A5: Security Misconfiguration

- **Description:** Insecure default configurations, incomplete or ad-hoc configurations, and failure to implement all security controls.
- **Example:** Leaving default admin passwords unchanged.
- **Mitigation:** Harden server configurations, disable unnecessary features, and ensure secure defaults.

A6: Vulnerable and Outdated Components

- **Description:** Using components with known vulnerabilities in an application.
- **Example:** Running an outdated version of a library with known security issues.
- **Mitigation:** Regularly update and patch software components, and monitor for known vulnerabilities.

A7: Identification and Authentication Failures

- **Description:** Weak authentication mechanisms that can be exploited by attackers.
- **Example:** Brute-force attacks on weak password policies.
- **Mitigation:** Implement multi-factor authentication (MFA) and enforce strong password policies.

A8: Software and Data Integrity Failures

- **Description:** Lack of integrity checks and controls over software and data.
- **Example:** Failure to verify the integrity of software updates.
- **Mitigation:** Use digital signatures, checksums, and secure update mechanisms.

A9: Security Logging and Monitoring Failures

- **Description:** Insufficient logging and monitoring can allow attackers to exploit vulnerabilities undetected.
- **Example:** Failing to log login attempts or suspicious activities.
- **Mitigation:** Implement comprehensive logging and monitoring, and regularly review logs for suspicious activities.

A10: Server-Side Request Forgery (SSRF)

- **Description:** Occurs when a web application is tricked into fetching a resource without validating the user-supplied URL.
- **Example:** An attacker sending requests to internal systems or other protected resources.

- **Mitigation:** Validate and sanitize user inputs, and restrict URL protocols and IP address ranges.

4. Best Practices for Web Security

- **Regularly Update and Patch Software:** Ensure that all components of your web application are up to date.
- **Use HTTPS:** Protect data in transit by using HTTPS to encrypt communications between the client and server.
- **Input Validation:** Never trust user input. Always validate and sanitize data before processing.
- **Output Encoding:** Encode output to prevent injection attacks such as XSS.
- **Access Control:** Implement role-based access control and enforce it at every layer of the application.

5. Conclusion

Understanding and mitigating the OWASP Top 10 vulnerabilities is a fundamental aspect of securing web applications. By adhering to best practices and regularly reviewing security controls, developers and security professionals can significantly reduce the risk of exploitation.