Security Testing Report and Validation

Security Testing Report and Validation 1
Introduction 2
1. Authentication and Authorization 2
1.1 Authentication Mechanisms 2
1.2 Authorization Mechanisms 2
1.3 Testing Results 2
2. Input Validation and Output Encoding 3
2.1 Input Validation 3
2.2 Output Encoding 3
2.3 Testing Results 3
3. Session Management 4
3.1 Mechanisms 4
3.2 Testing Results 4
4. Secure Data Transmission 5
4.1 Observations: 5
4.2 Recommendations for Future Implementation: 5
5. Error Handling and Logging 6
5.1 Observations: 6
5.2 Recommendations: 6
6. Third-Party Dependency Validation 7
6.1 Observations: 7
6.2 Recommendations: 7
7. Conclusion 8
Next Steps 8

Introduction

This document outlines the security testing and validation efforts for the UOW booking system. The goal is to ensure that the system adheres to best practices in application security, is resilient to common attacks, and fulfills its functional and security requirements.

1. Authentication and Authorization

1.1 Authentication Mechanisms

- Passwords are securely stored as salted and hashed values using `bcrypt`.
- Stateless sessions implemented using JSON Web Tokens (JWTs), with built-in expiration times to minimize token theft risks.

1.2 Authorization Mechanisms

- **Role-Based Access Control (RBAC):**
  - **Admin Role:** Manage rooms, view all bookings, and update system settings.
  - **User Role:** Book rooms, view bookings, and manage profiles.
- API routes enforce access control by verifying the JWT and its associated roles.

1.3 Testing Results

Test	Methodology	Result
Admin-only endpoints	Attempt access as a regular user	Access denied (403 Forbidden)
Regular user endpoints	Attempt access as an admin	Access allowed
Tampered JWT	Modify payload and use token	Access denied
Expired JWT	Attempt access with expired token	Access denied, re-authentication required
Missing JWT	Access protected route without token	Access denied (401 Unauthorized)

2. Input Validation and Output Encoding

2.1 Input Validation

- **Server-Side Validation:** All user input is validated against schema definitions.
- **Client-Side Validation:** Inputs are checked in real-time using regular expressions and form constraints.
- **Sanitization:** Special characters are escaped to prevent injection attacks.

2.2 Output Encoding

- All dynamic content is HTML-encoded before being displayed in the user interface to prevent Cross-Site Scripting (XSS) attacks.

2.3 Testing Results

Test	Payload/Methodology	Result
SQL Injection	`'; DROP TABLE users;--` as input	Input rejected
XSS Injection	`<script>alert('XSS')</script>` in form inputs	Script rendered as plain text
Invalid data formats	Submit incorrect email, date, or phone formats	Validation error messages shown
Boundary testing	Inputs exceeding field length constraints	Validation error messages shown

3. Session Management

3.1 Mechanisms

- JWT-based sessions ensure stateless authentication.
- JWT tokens include expiration settings, enforcing periodic re-authentication.
- Secure cookies and HTTP-only flags are applied when applicable.

3.2 Testing Results

Test	Methodology	Result
Token expiration	Use expired token for request	Access denied
Token replay attack	Attempt reusing the same token	Access denied on expiration
Secure cookie implementation	Validate HTTP-only and Secure flags	Flags properly set

4. Secure Data Transmission

Although HTTPS is not implemented in this student project, several best practices related to secure data transmission should be highlighted for future deployment in production environments. The current HTTP headers were analyzed, and no significant security risks were detected during testing. However, the following recommendations and observations should be noted:

4.1 Observations:

- **Current State**:
  - The application is served over HTTP.
  - API responses include headers such as `Vary: Origin`, `Access-Control-Allow-Credentials`, and caching controls like `Cache-Control: public`.
- **Testing Results**:
  - The `curl` command output indicates no use of HTTPS (`Strict-Transport-Security` header missing).
  - No sensitive data is transmitted over HTTP in this implementation since the application is locally hosted.

4.2 Recommendations for Future Implementation:

1. **Use HTTPS**:
   a. Obtain and configure an SSL/TLS certificate to serve all application resources securely.
   b. Enforce HTTPS with the `Strict-Transport-Security` (HSTS) header: `Strict-Transport-Security: max-age=31536000; includeSubDomains`

2. **Token Security**:
   a. Ensure all tokens (e.g., JWTs) are transmitted securely over HTTPS.
   b. Add a `Secure` flag to cookies if used.
3. **Verify Encryption**:
   a. Use strong encryption protocols like TLS 1.2+.
   b. Disable outdated protocols such as SSL 3.0 and TLS 1.0/1.1.

5. Error Handling and Logging

Error handling practices have been implemented thoroughly, ensuring no sensitive information is disclosed to users. However, logging mechanisms are currently absent and need to be added for robust application security.

5.1 Observations:

- **Current State**:
  - The application ensures generic error messages are shown to users.
  - Debug information (e.g., stack traces) is suppressed in production mode.
  - There is no centralized logging mechanism for server-side errors or user actions.
- **Testing Results**:
  - Manually induced errors returned secure and user-friendly error messages.
  - Logs are not yet collected for analysis or auditing.

5.2 Recommendations:

1. **Implement Logging Mechanism**:
   a. Use a logging library such as **Winston** or **Morgan** to capture application events.
   b. Include critical information in logs:
      i. Timestamps
      ii. Error types
      iii. HTTP request/response details (excluding sensitive data)
   c. Exclude sensitive data like passwords, tokens, or PII.
2. **Centralized Log Storage**:
   a. For production, store logs in a centralized system such as **ELK Stack** or **AWS CloudWatch**.
3. **Enable Alerts**:
   a. Set up alerts for critical events like failed login attempts or 500-level server errors.
4. **Testing and Monitoring**:
   a. Regularly review logs for anomalies.
   b. Conduct penetration testing to confirm error messages remain generic.

6. Third-Party Dependency Validation

The project relies on third-party dependencies, particularly in the Node.js ecosystem. No vulnerabilities were identified during dependency scans, but ongoing vigilance is necessary to ensure security over time.

6.1 Observations:

- **Current State**:
  - Dependency validation is performed using `npm audit`.
  - All dependencies are currently free from known vulnerabilities.
- **Testing Results**:
  - `npm audit` reported 0 vulnerabilities.
  - OWASP ZAP scans did not identify any significant issues with APIs.

6.2 Recommendations:

1. **Ongoing Dependency Management**:
   a. Regularly update dependencies and rerun `npm audit`.
   b. Enable automated tools like **Dependabot** or **Snyk** for continuous monitoring.
2. **Verify Trustworthiness**:
   a. Ensure dependencies are from reputable sources.
   b. Avoid unmaintained or deprecated packages.
3. **Use Lockfiles**:
   a. Use `package-lock.json` to lock dependency versions and prevent unexpected updates.
4. **Mitigate Supply Chain Risks**:
   a. Monitor for known vulnerabilities using tools like OWASP Dependency-Check or Retire.js.

7. Conclusion

The application demonstrates several good security practices, including:

- Proper error handling and suppression of sensitive debug information.
- Use of salted and hashed passwords.
- Thorough dependency management with no reported vulnerabilities.

Next Steps

1. **Implement Logging**: Add robust logging mechanisms to capture and analyze errors and anomalies.
2. **Migrate to HTTPS**: Secure data transmission by deploying the application with SSL/TLS in production.
3. **Continuous Testing**: Regularly perform dependency scans, run penetration tests, and monitor application logs for potential security breaches.
4. **Iterative Improvements**: As the application evolves, incorporate new security measures, such as enhanced RBAC, API rate-limiting, and advanced threat monitoring tools.

- End -