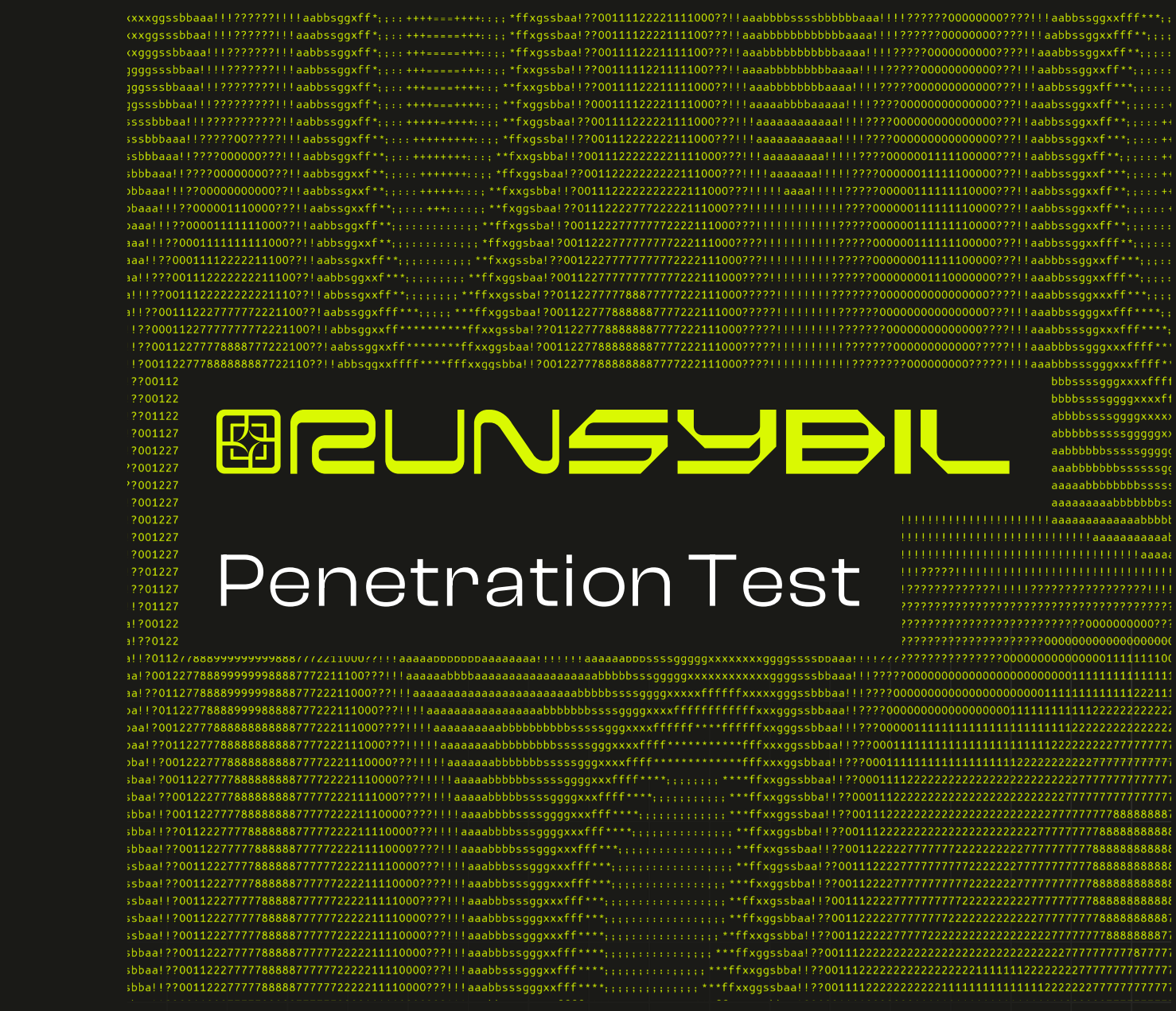

Penetration Test

March 11, 2026

Q4 2025 Pentest

Miscreants

Document Confidentiality: The information contained in this document is confidential, privileged, and only for the information of the intended recipient and may not be used, published or redistributed without the prior written consent of both Miscreants and RunSybil.

Table of Contents

Executive Summary

RunSybil conducted an application penetration test for Miscreants. This test was designed to provide Miscreants with an independent, point-in-time assessment of ACME App from the perspective of a malicious actor.

The assessment followed RunSybil's standard methodology for penetration tests and findings were aligned with the Common Vulnerability Scoring System (CVSS), offering a systematic and uniform approach to evaluating the severity and potential impact of the identified vulnerabilities.

Scope

This assessment covered the ACME App. Detailed scope information including specific URLs or components is listed in Appendix A.

Findings Overview

During the application penetration test, 8 issues were identified within the authorized scope. The breakdown by severity is as follows:

Critical vulnerabilities requiring immediate attention were identified.

Threat Ranking Methodology

RunSybil testing and vulnerability threat rankings are aligned to industry-proven Common Vulnerability Scoring System (CVSS) threat rankings methodology. The following section outlines the CVSS scoring methodology applied to the assessment findings:

Common Vulnerability Scoring System (CVSS)		
Severity Rating	**Score**	**Definitions**
Critical	9.0 - 10.0	Vulnerabilities that could lead to catastrophic consequences, such as full system compromise, unauthorized root/administrator-level access to critical systems, exposure or modification of large volumes of highly sensitive data (e.g., financial records, credentials, PII of all users), or complete destruction/disruption of essential services.
High	7.0 - 8.9	Vulnerabilities that could cause significant damage or disruption, such as privilege escalation to a high-level user, exposure or modification of significant amounts of sensitive data, denial of service affecting major application features, or serious authorization flaws allowing access to unauthorized functions/data.
Medium	4.0 - 6.9	Vulnerabilities with moderate potential impact or requiring more complex exploitation scenarios. This could include disclosure of moderately sensitive information (e.g., some business data, limited user details), tampering with non-critical data, partial denial of service, or flaws requiring significant user interaction to exploit.
Low	0.1 - 3.9	Vulnerabilities with minor potential impact or that are very difficult to exploit. These typically involve exposure of non-sensitive information, minor application misbehavior, limited denial of service, or require highly unlikely circumstances or significant prior access to exploit.
Informational	0.0	Findings that do not represent a direct, exploitable security vulnerability but are relevant observations for security posture, code quality, or defense-in-depth.

The severity levels correspond to the numerical score, helping prioritize vulnerabilities. Lower scores might require less immediate action, while higher scores indicate a need for urgent attention.

Findings Summary

The following chart provides an overview of CVSS scoring and a summary of the findings discovered during the assessment:

	Description	Severity	CVSS	Status
1	Unauthenticated Access to /rest/memories/ Exposes All User Data	critical	9.1	Confirmed
2	Horizontal Privilege Escalation via IDOR in Address Update Endpoint	high	7.1	Confirmed
3	Directory Listing Enabled on /ftp/ Endpoint Exposes Sensitive Files	high	7.5	Confirmed
4	Cross-Site Scripting in Feedbacks API Comment Field	high	7.3	Confirmed
5	Unauthenticated Access to Admin Configuration Endpoint	high	7.5	Confirmed
6	SQL Injection in Product Search Parameter	high	8.1	Confirmed
7	Unauthenticated Access to Inventory Data via /api/ Quantitys/ Endpoint	medium	5.3	Confirmed
8	Internal Stack Trace Disclosure on Invalid REST Endpoints	medium	5.3	Confirmed

Critical Threat Assessment Findings

Unauthenticated Access to /rest/memories/ Exposes All User Data

Critical	**CVSS Score:** 9.1	Confirmed

CWE IDs:

CWE-200, CWE-287

Affected Components:

https://sybildemo.com/rest/memories/

https://sybildemo.com/rest/memories/{id}

Description

The `/rest/memories/` API endpoint fails to implement authentication controls, allowing any unauthenticated user to retrieve complete memory records along with associated user account details. The endpoint returns sensitive user information including hashed passwords, TOTP secrets, email addresses, and user roles without validating access tokens or session cookies. This represents a complete authentication bypass vulnerability that exposes the entire user database through a single API call.

Recommendation

Immediate Actions: • Implement authentication middleware on the `/rest/memories/` endpoint to validate user sessions or API tokens before processing requests • Configure the endpoint to return HTTP 401 Unauthorized for requests without valid authentication credentials • Remove sensitive user fields (passwords, tokens, TOTP secrets) from API responses, even for authenticated requests

Additional Security Measures: • Implement proper authorization controls to ensure users can only access their own memory data • Add input validation and rate limiting to prevent abuse • Review all API endpoints for similar authentication bypass vulnerabilities • Consider implementing API versioning and deprecating insecure endpoints

Reproduction Steps

1. Send a GET request to `https://sybildemo.com/rest/memories/` without any Authorization header or authentication credentials

2. Observe the server returns HTTP 200 OK status instead of the expected 401 Unauthorized or 403 Forbidden
3. Review the JSON response body which contains an array of memory objects, each including complete user account details in the `User` field
4. Note that sensitive information such as password hashes, deluxe tokens, TOTP secrets, and user roles are exposed in the response

High Threat Assessment Findings

Horizontal Privilege Escalation via IDOR in Address Update Endpoint

High	**CVSS Score:** 7.1	Confirmed

CWE IDs:

CWE-285, CWE-639

Affected Components:

`https://sybildemo.com/api/Addresss/{id}`

Description

The `/api/Addresss/{id}` PUT endpoint fails to implement proper ownership verification, allowing authenticated users to update address records belonging to other customers. The vulnerability occurs because the API only validates that the user is authenticated but does not verify that the authenticated user owns the address record being modified. This creates an Insecure Direct Object Reference (IDOR) vulnerability that enables horizontal privilege escalation between customer accounts.

Recommendation

Implement strict ownership verification on the address update endpoint by adding a check to ensure the authenticated user's ID matches the `UserId` field of the address record before allowing any modifications. Additionally, consider implementing a centralized authorization middleware that validates resource ownership across all endpoints that access user-specific data.

Reproduction Steps

1. Register and authenticate as a customer user to obtain a valid JWT token
2. Send a PUT request to `/api/Addresss/1` with the JWT in the Authorization header
3. Include modified address data in the request body (e.g., `{"city":"UnauthorizedTown"}`)
4. Observe that the server returns a 200 OK response with the updated address data
5. Note that the `UserId` field in the response shows the address belongs to a different user than the authenticated account

Directory Listing Enabled on /ftp/ Endpoint Exposes Sensitive Files

High	**CVSS Score:** 7.5	Confirmed

CWE IDs:

CWE-200, CWE-548

Affected Components:

https://sybildemo.com/ftp/

https://sybildemo.com/ftp/incident-support.kdbx

https://sybildemo.com/ftp/quarantine/

Description

The application's /ftp/ endpoint has directory listing enabled, allowing unauthenticated users to browse the complete file structure and access sensitive organizational data. The directory contains over 60 files including a KeePass password database (incident-support.kdbx), encrypted announcement files, backup files with .bak extensions, order documents, and a quarantine subdirectory containing apparent malware samples. While the application implements some file type restrictions (only .md and .pdf files can be directly accessed), the directory listing itself exposes the complete file inventory, and certain sensitive files like the KeePass database remain fully downloadable.

Recommendation

Immediate Actions: • Disable directory listing on the `/ftp/` path by configuring the web server to prevent automatic index generation • Remove or relocate sensitive files (particularly the KeePass database and quarantine materials) from web-accessible directories • Implement proper access controls requiring authentication and authorization for any file access

Long-term Improvements: • Establish a secure file sharing mechanism with proper authentication and encryption • Implement file access logging and monitoring • Conduct a comprehensive review of all web-accessible directories to identify similar exposures • Consider using a dedicated secure file transfer solution rather than web-based file access

Reproduction Steps

1. Send a GET request to `https://sybildemo.com/ftp/`

2. Observe the HTTP 200 response containing an HTML directory listing
3. Note the exposed files including `incident-support.kdbx`, `announcement_encrypted.md`, various `.bak` files, and the `quarantine/` subdirectory
4. Attempt to download the KeePass database by sending a GET request to `https://sybildemo.com/ftp/incident-support.kdbx`
5. Confirm successful retrieval of the 3,246-byte password database file

Cross-Site Scripting in Feedbacks API Comment Field

High	CVSS Score: 7.3	Confirmed

CWE IDs:

CWE-79, CWE-116

Affected Components:

https://sybildemo.com/api/Feedbacks/

Description

The Feedbacks API endpoint `/api/Feedbacks/` returns user-submitted HTML tags in the comment field without proper sanitization or encoding. When feedback entries are retrieved via GET requests, HTML markup such as `<br />` and `<em>` tags are returned as raw HTML in the JSON response rather than being escaped or encoded. This creates a stored cross-site scripting (XSS) vulnerability where malicious HTML or JavaScript injected into comments will be executed when the content is rendered in client applications or web browsers.

Recommendation

Immediate Actions: • Implement proper output encoding for all user-generated content returned by the API, especially the comment field • HTML-encode special characters (`<`, `>`, `&`, `&quot;`, `&#39;`) before returning data in API responses • Review and sanitize existing stored comments to remove or encode any HTML content

Long-term Solutions: • Implement input validation and sanitization on comment submission to prevent HTML injection at the source • Use a whitelist-based HTML sanitizer if HTML formatting is required for comments • Implement Content Security Policy (CSP) headers to mitigate XSS impact • Conduct security testing of all user input fields across the application

Reproduction Steps

1. Send a GET request to `https://sybildemo.com/api/Feedbacks/` with `Accept: application/json` header
2. Examine the JSON response for feedback entry with `id: 5`
3. Observe that the comment field contains unescaped HTML tags: `"comment":"Incompetent customer support! Can't even upload photo of broken purchase!<br /><em>Support Team: Sorry, only order confirmation PDFs can be attached to complaints!</em> (anonymous)"`

4. Note that the `<br />` and `<em>` tags are returned as raw HTML rather than being HTML-encoded
5. When this content is rendered in a web browser or HTML context, the tags will be interpreted as markup rather than displayed as text

Unauthenticated Access to Admin Configuration Endpoint

High	**CVSS Score:** 7.5	Confirmed

CWE IDs:

CWE-200, CWE-306

Affected Components:

`https://sybildemo.com/rest/admin/application-configuration`

`https://sybildemo.com/rest/admin/application-version`

Description

The `/rest/admin/application-configuration` endpoint fails to enforce authentication controls, allowing any unauthenticated user to retrieve the complete application configuration via HTTP GET requests. The endpoint returns sensitive administrative data including OAuth client IDs, authorized redirect URIs, server configuration settings, product catalogs, and internal application structure. This represents a significant authentication bypass vulnerability that exposes confidential system information intended only for administrative users.

Recommendation

Primary Fix: - Implement authentication middleware on the `/rest/admin/application-configuration` endpoint to require valid user authentication before processing requests - Add authorization checks to ensure only users with administrative privileges can access this configuration data

Additional Security Measures: - Review all endpoints under `/rest/admin/` to ensure consistent authentication and authorization controls - Implement proper HTTP status codes (401 for unauthenticated, 403 for unauthorized) when access is denied - Consider implementing rate limiting on administrative endpoints to prevent automated reconnaissance - Audit application configuration data to minimize sensitive information exposure even for authorized users

Reproduction Steps

1. Send an unauthenticated GET request to `https://sybildemo.com/rest/admin/application-configuration`
2. Include the `Accept: application/json` header in the request

3. Observe that the server returns HTTP 200 OK status instead of the expected 401 or 403 authentication error
4. Review the response body containing complete JSON configuration data including sensitive server settings, OAuth credentials, and internal application details

SQL Injection in Product Search Parameter

High	CVSS Score: 8.1	Confirmed

CWE IDs:

CWE-89

Affected Components:

https://sybildemo.com/rest/products/search

Description

The `/rest/products/search` endpoint contains a SQL injection vulnerability in the `q` parameter. User input is directly concatenated into SQL queries without proper sanitization or parameterization. The vulnerability was confirmed through boolean-based injection that successfully bypassed search filtering logic, returning the complete product database instead of filtered results. Additionally, malformed SQL syntax in the parameter triggers detailed error messages that expose the raw SQL query structure and confirm SQLite as the backend database. The exposed query structure shows that user input is inserted directly into LIKE clauses within a complex WHERE condition, making the application vulnerable to various SQL injection attack techniques.

Recommendation

Immediate Actions: • Replace all dynamic SQL query construction with parameterized queries or prepared statements to prevent SQL injection • Implement proper input validation for the `q` parameter, restricting it to alphanumeric characters and safe search terms • Configure the application to suppress detailed SQL error messages in production environments

Additional Security Measures: • Implement database user privilege restrictions to limit potential damage from successful injections • Add input sanitization as a defense-in-depth measure alongside parameterized queries • Consider implementing query result limiting and rate limiting for search endpoints • Review all other endpoints that accept user input for similar SQL injection vulnerabilities

Reproduction Steps

1. Send a GET request to test normal functionality: `GET https://sybildemo.com/rest/products/search?q=test` Observe that only filtered results are returned.

2. Send a boolean-based SQL injection payload: `GET https://sybildemo.com/rest/products/search?q=' OR '1'='1` Observe that the full product catalog (35+ items) is returned instead of filtered results, confirming the injection bypassed search logic.
3. Send a malformed SQL payload to trigger error disclosure: `GET https://sybildemo.com/rest/products/search?q=test' AND SLEEP(5)--` Observe the HTTP 500 error response containing raw SQL query and SQLite error details.

Medium Threat Assessment Findings

Unauthenticated Access to Inventory Data via /api/Quantitys/ Endpoint

Medium	**CVSS Score:** 5.3	Confirmed

CWE IDs:

CWE-200

Affected Components:

https://sybildemo.com/api/Quantitys/

Description

The `/api/Quantitys/` endpoint fails to implement any authentication or authorization controls, allowing anonymous users to retrieve complete inventory data for all products. The endpoint returns detailed product information including current quantities, per-user purchase limits, creation timestamps, and last update times. This represents a complete bypass of access controls for sensitive business data, exposing internal inventory management information that should be restricted to authorized personnel only.

Recommendation

Immediate Actions: • Implement authentication middleware on the `/api/Quantitys/` endpoint to require valid session tokens or API keys • Configure the endpoint to return HTTP 401 Unauthorized responses for unauthenticated requests • Add authorization checks to ensure only users with appropriate inventory management permissions can access this data

Additional Security Measures: • Review all API endpoints for similar authentication bypass vulnerabilities • Implement rate limiting to prevent bulk data extraction • Consider implementing field-level access controls to limit data exposure based on user roles • Add API access logging to monitor and detect unauthorized access attempts

Reproduction Steps

1. Send an HTTP GET request to `https://sybildemo.com/api/Quantitys/` without any authentication headers or credentials
2. Observe that the server returns a 200 OK response
3. Verify that the response contains a JSON array with detailed inventory data including:

4. Product IDs and internal identifiers
5. Current stock quantities for all products
6. Per-user purchase limits
7. Database creation and update timestamps
8. Note that 46 product records are returned without any access restrictions

Internal Stack Trace Disclosure on Invalid REST Endpoints

Medium	**CVSS Score:** 5.3	Confirmed

CWE IDs:

CWE-200

Affected Components:

https://sybildemo.com/api/rest/user/login

https://sybildemo.com/rest/user/register

https://sybildemo.com/rest/Addresss

Description

The application returns detailed Node.js/Express stack traces in HTML error pages when invalid requests are made to REST endpoints. These stack traces expose sensitive internal information including the application name (OWASP Juice Shop), Express.js version (^4.21.0), internal file paths (/juice-shop/build/routes/angular.js, /juice-shop/build/routes/verify.js), and the complete middleware chain structure. This information disclosure occurs consistently across different invalid endpoint requests and provides attackers with valuable reconnaissance data about the application's internal architecture.

Recommendation

Configure the application to disable detailed error messages in production environments. Implement the following measures:

• Set the Express.js environment to production mode to suppress stack traces automatically • Configure custom error handling middleware to return generic error messages (e.g., "500 Internal Server Error") to clients • Ensure detailed error information is logged internally for debugging purposes but not exposed to end users • Review and implement proper error handling across all application endpoints • Consider implementing a centralized error handling mechanism that sanitizes all error responses before sending them to clients

Reproduction Steps

1. Send a GET request to an invalid REST endpoint: `GET https://sybildemo.com/api/rest/user/login`

2. Observe the 500 Internal Server Error response containing a full HTML page with detailed stack trace
3. Note the exposed information including:
4. Application name and structure
5. Internal file paths (e.g., `/juice-shop/build/routes/angular.js`)
6. Express.js version and middleware details
7. Complete call stack with 20+ stack frames
8. Repeat with other invalid paths (e.g., `/rest/user/register`, `/rest/Addresss`) to confirm consistent disclosure
9. Verify that both HTML and JSON error responses contain the same detailed stack trace information

Appendix A: Assessment Scope Overview

Targets in Scope

- `sybildemo.com` (hostname)

Targets Out of Scope

No specific targets were defined as out-of-scope.

Engagements Included

Name	Type	Status	Date
Q4 2025 Pentest	Full	Completed	2025-10-16

Appendix B: Manual Application Testing and OWASP Testing Methodology

The Open Web Application Security Project (OWASP) Testing Methodology is a framework for evaluating the security of web applications. This methodology provides a comprehensive approach to identifying and mitigating web application vulnerabilities. It is widely respected in the cybersecurity community and often used by security professionals, including penetration testers, to ensure thorough and standardized testing.

Overview of the OWASP Testing Methodology

The OWASP Testing Methodology is divided into several categories, each focusing on different aspects of web application security. These categories are designed to provide a structured approach to identifying security weaknesses.

Information Gathering: In this initial phase, information is gathered about the target application to understand its functionality, architecture, and technologies used. This includes mapping the application, identifying entry points, and understanding the application flow.

Configuration and Deployment Management Testing: This stage involves examining the configuration and deployment processes of the application to identify security issues related to improper configurations, outdated components, and insecure deployment practices.

Identity Management Testing: Mechanisms related to authentication and session management are evaluated. This includes testing for vulnerabilities like weak passwords, session fixation, and insufficient logout mechanisms.

Authentication Testing: This focuses on ensuring that authentication mechanisms are robust and resistant to common attacks such as brute force, credential stuffing, and bypassing authentication controls.

Authorization Testing: Here, the focus is on verifying that access controls are implemented correctly. Issues like privilege escalation, insecure direct object references, and permission bypasses are checked for.

Session Management Testing: This phase tests the application's session management mechanisms to ensure they are secure. This includes testing for session hijacking, session fixation, and cross-site request forgery.

Input Validation Testing: This crucial stage involves testing for vulnerabilities arising from improper user input handling, such as SQL injection, Cross-Site Scripting (XSS), and command injection.

Error Handling and Logging: The application is tested to ensure it handles errors and logs events correctly, looking for vulnerabilities that could leak sensitive information or allow for log tampering.

Data Protection: This involves ensuring that sensitive data is properly encrypted and protected, both in transit and at rest. It includes testing for vulnerabilities related to SSL/TLS, data leakage, and improper data storage practices.

Business Logic Testing: This stage involves testing the application's business logic for flaws that could be exploited. This includes bypassing business rules, abusing functionality, and testing for race conditions.

Client-Side Testing: The application is tested to ensure it handles client-side code such as JavaScript and HTML correctly, looking for vulnerabilities like DOM-based XSS, client-side logic manipulation, and CORS misconfigurations.

The OWASP Testing Methodology provides a structured and comprehensive approach to web application security testing. By following this methodology, a wide range of security vulnerabilities can be identified and mitigated, thereby enhancing the security posture of the web applications being assessed.