haiderm Security
haiderm.com

Web Application Security Assessment Report

INDEPENDENT SECURITY ASSESSMENT

PREPARED BY	haiderm Security
WEBSITE	https://haiderm.com
CLIENT WEBSITE	example-client-site.com

Confidentia

01 | DOCUMENT CONTROL

Field	Value
Document Title	Website and API Penetration Testing Report
Version	1.0
Date	mm/dd/yy
Author	HaiderM Security
Status	Client-facing assessment deliverable
Classification	Confidential
Distribution	Prospective client evaluation
Credentials	CISSP, ISSAP, OSCP, MSc Cyber Security, 10+ years of cyber security experience

02 | EXECUTIVE SUMMARY

This sample assessment reviews the public-facing website and API surface at example-client-site.com and demonstrates how a formal engagement would present application security risk, business impact, evidence, remediation steps, and retest guidance for a client-facing decision maker.

Coverage includes web application mapping, API endpoint review, authentication and session handling, authorization controls, input validation, business logic checks, security headers, TLS posture, file and directory exposure, sensitive data exposure and safe unauthenticated web/API testing.

The overall risk rating is Critical because two authorization and API control issues could plausibly lead to account takeover or sensitive data compromise if confirmed in a live engagement. The report contains 9 findings: 2 Critical, 3 High, 2 Medium, 1 Low, and 1 Informational. The five immediate priorities are the 2 Critical and 3 High findings: fix broken access control, review privileged API actions, add rate limiting, strengthen authentication controls, and retest after remediation.

This sanitised sample avoids naming a real target system or demonstrating exploit payloads. In a live client report, Critical and High findings would include exact affected endpoints, roles, parameters, timestamps, scoped request and response evidence, screenshots when useful, and retest proof after remediation.

OVERALL RISK	TOTAL FINDINGS	IMMEDIATE PRIORITIES	RETEST ADVICE
Critical	**9**	**5**	**Yes**
Immediate remediation is required for the two Critical issues.	2 Critical | 3 High | 2 Medium | 1 Low | 1 Info	2 Critical + 3 High findings require action in the first 7 days.	Targeted validation after remediation.

TOP RISK AREA	PRIMARY EXPOSURE	BEST FIRST ACTION	REMEDIATION PACE
Broken access control	**API attack surface**	**Fix access control**	**3 days**
Authorization flaws across web and API workflows are the most urgent remediation area.	Public endpoints, weak authorization boundaries and risky workflows increase targeted attack likelihood.	Correct object-level and function-level authorization before lower-risk hardening work.	Critical and High actions should start immediately, with retesting after fixes.

03 | RISK RATING

Severity	Count	Meaning
Critical	2	Immediate, high-impact compromise risk with realistic paths to site takeover, code execution, administrative compromise or sensitive data exposure.
High	3	Significant risk with realistic exploitation paths and strong potential for account compromise, service abuse or major control failure.

Medium	2	Important weakness that increases exposure or contributes meaningfully to attack success, but is less immediately damaging on its own.
Low	1	Limited direct impact, often relating to information leakage or smaller hardening gaps.
Informational	1	Governance or defence-in-depth observation without direct exploitability in the current context.

04 | SCOPE

In scope	Out of scope
• Public website and API endpoints • Authentication, session and access-control checks • API object-level and function-level authorization • Version disclosure checks • Security headers and TLS • Sensitive data exposure review • Rate limiting and anti-automation controls • File and directory exposure • Technology fingerprinting and exposed file checks • Non-destructive business logic and input validation review	• Denial of service • Social engineering or phishing • Live credential stuffing or password spraying against real accounts • Physical security • Destructive exploitation • Production data modification • Admin panel testing unless credentials are provided • Source code review unless separately agreed

05 | POSITIVE OBSERVATIONS

- HTTPS is enabled
- No obvious public stack trace or framework debug page was observed.
- No obvious public data dump or backup exposure was observed.
- No public database error leakage was identified.
- Administrative and privileged routes do not appear intentionally indexed.
- No sensitive payment data was identified during unauthenticated testing.
- Public content did not expose obvious customer records.
- Contact forms did not show obvious reflected handling issues during safe testing.

06 | FINDINGS SUMMARY

ID	Finding	Severity	Affected area	Priority
C-01	Broken Object Level Authorization Allows Unauthorized Data Access	Critical	API object and record access controls	Immediate
C-02	Broken Function Level Authorization Allows Privileged Action Abuse	Critical	Privileged web/API workflow	Immediate
H-01	Missing API Rate Limiting and Anti-Automation Controls	High	Authentication and API endpoints	Immediate
H-02	Weak Authentication and Session Management Controls	High	Login, password reset and session handling	Immediate
H-03	Injection Risk in Web/API Input Handling	High	Public request parameters and API input	Immediate
M-01	Excessive Data Exposure in API Responses	Medium	API response bodies and client-side data	High
M-02	Missing or Weak Security Headers	Medium	HTTP response headers	High
L-01	Technology and Version Disclosure	Low	HTML source or asset query strings	Medium

I-01	No Web Application Firewall or API Protection Detected	Informational	Edge protection	Planned

07 | RETEST STATUS

ID	Severity	Retest status	Closure evidence expected
C-01	Critical	Pending remediation	Authorization checks enforced server-side and clean external validation completed.
C-02	Critical	Pending remediation	Privileged workflow fixed, role boundaries reviewed and no unauthorized action path remains.
H-01	High	Pending remediation	Rate limiting, throttling or challenge behaviour confirmed on exposed endpoints.
H-02	High	Pending remediation	MFA, session controls and lockout/challenge behaviour confirmed.
H-03	High	Pending remediation	Patched input-handling path and safe validation of database or command interaction.
M-01	Medium	Pending remediation	API responses reduced to intended fields or accepted with documented business need.
M-02	Medium	Pending remediation	Security-header baseline present without breaking site functionality.
L-01	Low	Pending remediation	Avoidable technology and version disclosure reduced where practical.
I-01	Informational	Client decision	WAF/CDN control accepted, deferred or implemented with logging enabled.

08 | FINDINGS

C-01 | Broken Object Level Authorization Allows Unauthorized Data Access	CRITICAL
AFFECTED COMPONENT	API object and record access controls
BUSINESS IMPACT	Unauthorized access to customer records, account data or business objects
PRIORITY	Immediate
SCORE	9.8
DESCRIPTION	In this sample scenario, an API endpoint is treated as Critical because object identifiers or record references could be changed to access data belonging to another user or tenant. Broken Object Level Authorization is one of the highest impact API risk classes because successful exploitation may expose customer data, support account takeover, bypass tenancy boundaries and undermine trust in the application.
TECHNICAL EVIDENCE	`Sample evidence would include the affected endpoint, HTTP method, object identifier, authenticated user role, timestamped request/response evidence and safe validation showing that one user can access another user's object. This sample report intentionally uses sanitised evidence and does not demonstrate exploit payloads.`

REMEDIATION	Treat this as an emergency authorization defect. Preserve relevant logs, identify the affected object types and confirm whether access crossed user, tenant or role boundaries before making broad code changes. Enforce server-side object ownership and tenancy checks on every affected endpoint. Do not rely on hidden fields, client-side route controls, predictable IDs or UI visibility to protect records. Replace direct object lookup patterns with scoped queries that bind the requested object to the authenticated user, tenant, organization and role. Deny access by default where the relationship cannot be proven. Review related read, update, delete, export and bulk endpoints because the same authorization mistake often appears across a family of routes. Assume sensitive data may have been accessed if logs show repeated object ID changes, cross-account requests, unusual export activity or high-volume API reads. Rotate affected credentials or tokens if account misuse is suspected. Verify the fix by retesting with at least two users from different roles or tenants. Requests for unauthorized objects should return a controlled 403 or equivalent denial without leaking record details.
RETEST GUIDANCE	Confirm object-level authorization is enforced server-side across read, write, delete and export paths, then re-check for predictable identifier abuse.
REFERENCES	OWASP API Security Top 10 - https://owasp.org/API-Security/ OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/ OWASP Web Security Testing Guide - https://owasp.org/www-project-web-security-testing-guide/

C-02 | Broken Function Level Authorization Allows Privileged Action Abuse	**CRITICAL**
AFFECTED COMPONENT	Privileged web/API workflow
BUSINESS IMPACT	Unauthorized privileged actions and account or data compromise
PRIORITY	Immediate
SCORE	9.4
DESCRIPTION	In this sample scenario, a privileged workflow is treated as Critical because an authenticated low-privileged user could invoke an administrative or high-impact action directly through the web application or API. Function-level authorization failures can allow users to change roles, approve transactions, modify security settings, access exports or perform actions that should be reserved for trusted staff.
TECHNICAL EVIDENCE	`Sample evidence would include the affected route, HTTP method, expected role, actual role used during testing, relevant request parameters and safe validation showing whether a lower-privileged account can perform the action. No destructive privilege escalation is performed in this sample.`
REMEDIATION	Apply a server-side authorization fix immediately for the affected function. The route, controller, resolver, background action and any API gateway policy should all enforce the same role and permission requirement. Temporarily disable or restrict the affected workflow until the fix is deployed. Where the workflow is business-critical, limit access to named trusted users and increase monitoring until retesting is complete. Audit recent privileged actions after patching. Look for role changes, unfamiliar admins, unexpected exports, configuration changes, payment or order modifications and activity around suspicious timestamps. Enforce MFA for privileged users, remove shared administrator accounts, require named individual accounts and limit administrative access by trusted network or identity-aware access where possible. Review route-level guards, API gateway policies, GraphQL resolvers, background job triggers and client-side-only checks. Confirm users cannot influence role, tenant or permission decisions through request parameters or metadata. Retest with accounts representing each role. A low-privileged account should not be able to invoke, queue, replay or indirectly trigger privileged actions.

RETEST GUIDANCE	Verify that privilege boundaries are correctly enforced and that privileged actions cannot be invoked through direct API calls, modified requests or hidden UI routes.
REFERENCES	OWASP API Security Top 10 - https://owasp.org/API-Security/ OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/ OWASP Web Security Testing Guide - https://owasp.org/www-project-web-security-testing-guide/

H-01 | Missing API Rate Limiting and Anti-Automation Controls	HIGH
AFFECTED COMPONENT	Authentication and API endpoints
BUSINESS IMPACT	Automated abuse, account attack and resource exhaustion risk
PRIORITY	Immediate
SCORE	8.1
DESCRIPTION	The target exposes login, password reset or API workflows without visible evidence of strong rate limiting, throttling, bot management or equivalent anti-automation controls. This becomes High risk when endpoints can be repeatedly called to guess credentials, enumerate accounts, abuse one-time codes, scrape data or consume backend resources.
TECHNICAL EVIDENCE	`POST /api/auth/login HTTP/1.1 Host: example-client-site.com HTTP/1.1 401 Unauthorized No rate-limit header, lockout signal, increasing delay or challenge response was observed during safe low-volume testing. A live engagement would validate this with an agreed test plan and without credential stuffing.`
REMEDIATION	Apply rate limiting to authentication, password reset, registration, search, export and high-cost API endpoints. Limits should consider IP address, account identifier, session, token, device fingerprint and tenant where appropriate. Use progressive controls rather than a single global block. Add throttling, short cooldowns, CAPTCHA or proof-of-work challenges for suspicious patterns, and stricter limits for unauthenticated or newly created sessions. Protect account recovery and one-time-code workflows separately. Limit resend, verification and password-reset attempts, and avoid responses that confirm whether an account exists. Monitor for repeated failed logins, high-volume API calls, scraping patterns, token guessing, unusual countries or hosting-provider source networks. Feed these events into the WAF, API gateway or SIEM for automated blocking and alerting. Verify the fix from an external network. Repeated requests should trigger clear throttling, challenge or lockout behaviour while normal user journeys remain usable.
RETEST GUIDANCE	Confirm exposed endpoints are rate-limited or challenged and that abusive request patterns are logged.
REFERENCES	OWASP API Security Top 10 - https://owasp.org/API-Security/ OWASP Automated Threats to Web Applications - https://owasp.org/www-project-automated-threats-to-web-applications/ OWASP Web Security Testing Guide - https://owasp.org/www-project-web-security-testing-guide/

H-02 | Weak Authentication and Session Management Controls	HIGH
AFFECTED COMPONENT	Login, password reset and session handling
BUSINESS IMPACT	Higher likelihood of account compromise and session misuse

PRIORITY	Immediate
SCORE	7.6
DESCRIPTION	The application authentication surface appears directly reachable without visible evidence of strong MFA enforcement, robust session expiry, secure cookie settings or equivalent protection for privileged users.
TECHNICAL EVIDENCE	`$ curl -I https://example-client-site.com/login HTTP/1.1 200 OK Set-Cookie: session=sample; No visible MFA requirement, secure session policy evidence or anti-automation response was observed during safe unauthenticated review. A live engagement would validate this with agreed test accounts.`
REMEDIATION	Require MFA for administrators, finance users and other privileged roles. Prefer authenticator-app, passkey or hardware-key MFA over email-only codes where possible. Harden session cookies with Secure, HttpOnly and SameSite attributes. Regenerate session identifiers after login and privilege changes, and expire sessions after appropriate idle and absolute timeouts. Reduce the number of high-value targets. Remove unused privileged accounts, disable shared accounts and require each privileged user to have a named individual account. Improve password and recovery controls. Require long unique passwords, block known compromised passwords, protect reset flows from enumeration and ensure admin email accounts are also MFA-protected. Add layered blocking through a WAF, identity provider, API gateway or application control to challenge repeated login attempts and alert on unusual authentication activity. Verify by using controlled test accounts: failed attempts should trigger throttling or challenge behaviour, privileged access should require MFA, and session cookies should carry the expected security attributes.
RETEST GUIDANCE	Confirm anti-automation, MFA and secure session controls trigger appropriately for privileged access.
REFERENCES	OWASP Top 10 A07 Identification and Authentication Failures - https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/ OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/ OWASP Web Security Testing Guide - https://owasp.org/www-project-web-security-testing-guide/

H-03 | Injection Risk in Web/API Input Handling	**HIGH**
AFFECTED COMPONENT	Public request parameters and API input
BUSINESS IMPACT	Potential database compromise, command execution or sensitive data exposure
PRIORITY	Immediate
SCORE	8.6
DESCRIPTION	The assessment identified exposed web or API input handling that should be treated as a high-priority injection risk until the affected code path is patched or safely ruled out. Unsafe handling of request parameters can allow attackers to alter SQL, NoSQL, OS command, template, LDAP or server-side request behaviour depending on the backend implementation.
TECHNICAL EVIDENCE	`Public parameter handling patterns indicate that backend-facing functionality requires immediate secure code validation. Safe validation should focus on parameterised query handling, controlled error behaviour, time-delay indicators in a safe environment and affected endpoint version or deployment checks. No destructive exploit traffic was performed in this sample report.`

REMEDIATION	Patch the affected endpoint, controller or service immediately. If the vulnerable functionality is custom-built, route the fix through code review and deploy only after testing on a staging copy of the application. Replace unsafe backend calls with parameterized queries, safe ORM patterns, command argument separation and strict input validation. Avoid concatenating request values into SQL, NoSQL filters, shell commands, templates or server-side URLs. Apply allow-lists for IDs, sort fields, filters, operators, filenames, redirect targets and enum values. Reject unexpected structure rather than trying to clean arbitrary input after the fact. Reduce exposure while the permanent fix is prepared. Disable the affected feature, restrict it to authenticated users, add WAF rules for injection patterns and monitor requests containing unusual quotes, comments, encodings, template syntax or time-delay payloads. Check whether data may have been accessed or altered. Review database users, recent admin activity, suspicious content changes, unexpected exports, background jobs and new application accounts. Verify the fix with safe retesting: malicious-looking input should be rejected or parameterized, backend errors should not be exposed, and the application should return controlled responses without changing query or command structure.
RETEST GUIDANCE	Re-check the affected workflow after patching and validate that user-controlled input cannot alter backend query, command, template or request structure.
REFERENCES	OWASP Injection Prevention Cheat Sheet - https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html OWASP SQL Injection - https://owasp.org/www-community/attacks/SQL_Injection PortSwigger SQL Injection Academy - https://portswigger.net/web-security/sql-injection

M-01 | Excessive Data Exposure in API Responses	**MEDIUM**
AFFECTED COMPONENT	API response bodies and client-side data
BUSINESS IMPACT	Improves attacker insight and may expose sensitive business or user data
PRIORITY	High
SCORE	5.3
DESCRIPTION	The public API appears to return more data than the client requires for the tested workflow. Excessive data exposure is common where APIs return full objects and rely on the front end to hide fields. The risk becomes more meaningful when responses include internal IDs, roles, email addresses, tokens, account status, pricing logic or other fields useful for attack planning.
TECHNICAL EVIDENCE	`GET /api/users/me HTTP/1.1 200 OK` `{"id":1,"email":"user@example.com","role":"customer","internalFlags":["sample"]}`
REMEDIATION	Review each API response against the user journey that consumes it. Remove fields that are not needed by the client and avoid returning internal metadata by default. Use explicit response DTOs, serializers or schema allow-lists rather than returning database models directly. Sensitive fields should require a clear business reason and role check before release. Treat identifiers, roles and workflow status fields as useful attacker context even where they are not secrets. Pair necessary exposure with MFA, access control, rate limiting and monitoring. Check REST, GraphQL and mobile API endpoints for over-broad objects, nested relations, debug fields, stack traces, feature flags and hidden administrative state. Verify by requesting representative API endpoints as unauthenticated, low-privileged and privileged users. Responses should contain only fields intentionally required for that role and workflow.
RETEST GUIDANCE	Re-check public and authenticated API responses for unnecessary fields and role-specific leakage.

REFERENCES	OWASP API Security Top 10 - https://owasp.org/API-Security/ OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/ OWASP Web Security Testing Guide - https://owasp.org/www-project-web-security-testing-guide/

M-02 | Missing or Weak Security Headers	MEDIUM
AFFECTED COMPONENT	HTTP response headers
BUSINESS IMPACT	Higher exposure to clickjacking and content abuse
PRIORITY	High
SCORE	6.1
DESCRIPTION	The response set lacks a stronger baseline of modern security headers such as Content-Security-Policy, frame protections, Referrer-Policy and Permissions-Policy.

TECHNICAL EVIDENCE	`$ curl -I https://example-client-site.com/`
	`HTTP/1.1 200 OK`
	`Content-Security-Policy: not present`
	`X-Frame-Options: not present`
	`Referrer-Policy: not present`

REMEDIATION	Define a security-header baseline for the site and apply it at the web server, CDN or application gateway layer so it covers application pages, API responses, static assets and error responses consistently. Add or tune Content-Security-Policy in report-only mode first. Inventory required script, style, image, font, frame and connection sources, then move to enforcement once legitimate site functionality is confirmed. Set clickjacking protection with frame-ancestors in CSP, or X-Frame-Options if legacy support is needed. Only allow trusted framing partners when the business has a clear requirement. Add Referrer-Policy, Permissions-Policy, X-Content-Type-Options and HSTS where appropriate. For HSTS, confirm HTTPS works correctly across the main domain and subdomains before enabling long max-age or includeSubDomains. Test the headers on the homepage, login page, API responses, redirects and representative content pages. Confirm the changes do not break forms, embedded media, analytics, single-page application behaviour or third-party integrations. Maintain the header configuration as part of deployment documentation so future application, CDN or hosting changes do not silently remove the controls.
RETEST GUIDANCE	Validate the new header baseline and ensure the policy set does not break required front-end functionality.
REFERENCES	OWASP Secure Headers Project - https://owasp.org/www-project-secure-headers/ Mozilla HTTP Observatory - https://developer.mozilla.org/en-US/observatory MDN Content-Security-Policy - https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

L-01 | Technology and Version Disclosure	LOW
AFFECTED COMPONENT	HTML source or asset query strings
BUSINESS IMPACT	Supports environment profiling
PRIORITY	Medium
SCORE	3.1
DESCRIPTION	Version information appears inferable from source output or asset version parameters. This can help attackers narrow exploit research more quickly.
TECHNICAL EVIDENCE	`Observed HTML and static asset references included version-style query parameters.`

REMEDIATION	Keep frameworks, libraries, runtimes and server components fully patched; this is more important than hiding versions. Establish a regular update cadence with backups, staging checks and rollback steps. Reduce avoidable version leakage from generator tags, response headers, error pages, asset query strings, public manifests and exposed metadata where doing so does not break cache-busting or functionality. Review frontend bundles, JavaScript source maps, package manifests, API documentation and build artefacts for hard-coded version comments or public changelog references. Pair this with inventory management. Maintain a list of installed component versions internally so defenders can patch quickly without needing public disclosure to identify exposure. Verify by checking page source, public metadata files and common scanner-visible locations. Remaining version information should either be intentionally public or operationally necessary.
RETEST GUIDANCE	Review HTML and asset paths again after hardening changes.
REFERENCES	OWASP WSTG Information Gathering - https://owasp.org/www-project-web-security-testing-guide/ OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/

I-01 | No Web Application Firewall or API Protection Detected	INFO
AFFECTED COMPONENT	Edge protection
BUSINESS IMPACT	Reduced resilience against automated web/API attack traffic and commodity exploitation attempts
PRIORITY	Planned
SCORE	0.0
DESCRIPTION	No clear public signal of a dedicated web application firewall, API gateway security policy, bot control or equivalent edge protection was observed. This is not a confirmed vulnerability and should not be treated as mandatory by itself; it is included as a defence-in-depth improvement for internet-facing websites and APIs exposed to automated scanning and opportunistic exploitation.
TECHNICAL EVIDENCE	`Public response headers and edge behaviour did not clearly indicate a WAF challenge, managed security CDN, API gateway policy or application-layer filtering control. Internal confirmation with the client or hosting provider would be required before treating this as definitive.`
REMEDIATION	Select an edge protection option appropriate for the application, such as a managed WAF, API gateway security policy, bot management control or security-enabled CDN. Enable managed web rules, API schema-aware validation where available, bot protections, authentication endpoint controls and virtual patching for known framework or library vulnerabilities. Start in monitoring or low-friction mode if the application has checkout, forms or customer portals, then move to stricter blocking once false positives are reviewed. Configure alerting for blocked exploit attempts, repeated login failures, scraping and suspicious API request patterns.
RETEST GUIDANCE	Confirm that the selected control is active on the production domain, that common web/API attack patterns are logged or blocked, and that normal site functions such as forms, login, checkout and API integrations remain usable.
REFERENCES	OWASP ModSecurity Core Rule Set - https://owasp.org/www-project-modsecurity-core-rule-set/ OWASP API Security Top 10 - https://owasp.org/API-Security/ OWASP WSTG - https://owasp.org/www-project-web-security-testing-guide/

09 | COVERAGE CHECKLIST

Control area	Status	Observation
Application and API inventory	Observed	Public routes and API endpoints should be mapped so exposed functionality, authentication requirements and data flows are understood before closure.
Access-control exposure	Priority	Critical authorization exposure is the leading remediation area; affected endpoints, roles and object relationships should be mapped before closure.
Framework and library exposure	Improvement	Frontend and backend component versions can support fingerprinting; unused packages and exposed build artefacts should be removed.
Authentication security	Priority	Login protection should include rate limiting, MFA, secure session handling, strong password policy and alerting for repeated failed attempts.
MFA recommendation	Recommended	MFA should be mandatory for administrators and privileged users, with recovery codes stored securely and shared accounts removed where possible.
API rate limiting	Priority	Public API access should be rate-limited and monitored, especially for login, password reset, search, export and high-cost endpoints.
REST API	Improvement	Unauthenticated and low-privileged responses should be reduced where feasible, especially where response fields support targeting or data harvesting.

Security headers	Improvement	A baseline should include CSP planning, frame protection, Referrer-Policy, Permissions-Policy, X-Content-Type-Options and HSTS where suitable.
TLS	Good	HTTPS is enabled; confirm certificate chain quality, protocol support, redirect behaviour and HSTS readiness during retesting.
Directory listing	Checked	No directory-listing finding is retained, but uploads, cache, temporary and static asset paths should still be spot-checked after deployments.
Backups	Checked	No exposed-backup finding is retained, but backup and export files should be stored outside the web root and covered by routine exposure checks.
Debug files	No issue	No public debug leakage was confirmed; keep production debug mode disabled and ensure logs are not written to public paths.
Malware indicators	No issue	No obvious public compromise indicators were observed; this does not replace authenticated file-integrity review or server-side malware scanning.
Blacklist indicators	No issue	No immediate blacklist warning behaviour was seen; monitor search-console, safe-browsing and email reputation signals after remediation.
WAF or CDN	Opportunity	No clear WAF or API protection signal was detected; a managed WAF, API gateway policy or security-enabled CDN would add virtual patching, bot filtering and better attack visibility.
Logging and monitoring	Review needed	Client-side confirmation is needed for access logs, security alerts, administrator activity logs and a process for reviewing high-risk events.
Contact forms	No issue	No obvious reflected handling issue was observed during safe testing; forms should still be protected with spam controls, validation and alerting.

10 | REMEDIATION ROADMAP

EMERGENCY CONTAINMENT
0 TO 12 HOURS

- Fix critical broken access-control paths.
- Review privileged accounts and recent role or permission changes.
- Restrict or rate-limit exposed API endpoints that can be abused.
- Enable MFA, secure session controls and login protection for privileged users.
- Preserve logs and evidence before major cleanup.

CONTROL HARDENING
12 TO 48 HOURS

- Validate and patch injection exposure
- Reduce excessive API data exposure where feasible
- Deploy a security-header baseline
- Decide on WAF/CDN protection and enable logging
- Reduce avoidable version disclosure

VERIFICATION
48 TO 72 HOURS

- Retest all Critical and High findings
- Document accepted residual risks
- Create application ownership and dependency update policy
- Improve monitoring and alert review
- Schedule periodic website and API security reassessment

11 | WHAT HAIDERM SECURITY WOULD DO NEXT

After delivering this assessment, the next stage would focus on safe validation, practical remediation support, and closure evidence. The goal is not just to identify issues, but to help the client reduce risk quickly and prove the fixes worked.

Step	Outcome
Validate affected components	Confirm exact affected endpoints, roles, parameters, advisories and real exposure within the agreed scope.
Support remediation	Help prioritise code fixes, configuration changes, compensating controls and deployment sequencing without disrupting normal operations.
Retest fixes	Re-check remediated findings and provide clear fixed, partially fixed or still exposed status.

Step	Outcome
Close the loop	Provide a concise closure note that the client can keep for governance, insurer, stakeholder or supplier assurance needs.

12 | TOOLS AND TECHNIQUES

Assessment area	Tools/scripts
Manual request validation	Burp Suite, OWASP ZAP, browser DevTools
Web and API attack-surface mapping	ProjectDiscovery httpx, katana, naabu, Wappalyzer, certificate transparency review
Known-vulnerability checks	Nuclei with curated safe templates, OSV, GitHub Advisories, vendor advisories
HTTP header and policy review	Mozilla HTTP Observatory, ProjectDiscovery httpx, curl header script
TLS and certificate review	SSL Labs Server Test, testssl.sh
Passive reconnaissance	OWASP Amass, certificate transparency review, httpx reachability checks
Retest and closure evidence	Burp Repeater, curl replay script, screenshot comparison, response-diff script
Authenticated API and workflow review	Burp Suite, Postman or Bruno, jwt_tool, custom authorization matrix script

13 | OFFICIAL HARDENING GUIDANCE REFERENCES

- OWASP API Security Top 10 - https://owasp.org/API-Security/
- OWASP Web Security Testing Guide (stable) https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/
- OWASP Application Security Verification Standard - https://owasp.org/www-project-application-security-verification-standard/
- NCSC Application Development Guidance - https://www.ncsc.gov.uk/guidance/application-development-guidance-introduction

14 | ABOUT HAIDER QURESHI

Haider Qureshi is an application security professional based in London, with more than 10 years of cyber security experience across banking, telecoms, technology, and consulting. His work focuses on secure SDLC, threat modelling, secure design reviews, penetration testing, cloud security, and practical remediation support.

Haider holds CISSP, ISSAP, and OSCP certifications, and an MSc in Cyber Security from Royal Holloway, University of London. His approach combines attacker-aware technical testing with clear business-risk reporting, so clients understand what matters, why it matters, and how to fix it safely.

15 | WORK WITH HAIDER

If you need a website and API penetration test, remediation support, or retesting after fixes, you can contact Haider directly using the links below. Engagements can be scoped around public web/API exposure, authentication and authorization, business logic, input validation, security headers, TLS, sensitive data exposure and clear remediation reporting.

Contact: https://haiderm.com/contact/