Fencer

Secure software, **early**.

HONEY CORP

# Penetration Test Remediation Report

Version 1.0

honey-corp.com Fencer Agent Penetration Test - Aug 20, 2026: August 20, 2026 - August 20, 2026

# Table of Contents

# 1 Legal Responsibility

The contents of this report are **CONFIDENTIAL** and must not be disclosed or reproduced in any form to third parties, whether in hardcopy or electronic format (softcopy), without the express written consent of both parties. Upon delivery of this report to **Fencer**, the right to reproduce, share with third parties, and distribute it falls under the discretion and responsibility of your institution.

It should be noted, however, that sharing, duplicating, or distributing this report, either internally or externally, could pose significant risks and damage to your cybersecurity and information systems. **Fencer** shall not be held liable for any risks or losses incurred due to the sharing, duplication, and distribution actions taken by your institution, or as a result of these actions.

This report reflects the status of the systems within the defined scope in relation to known vulnerabilities at the time of testing. The testing period was designed to identify and document the maximum number of vulnerabilities. However, attackers with more time, resources, and advanced capabilities might discover and exploit vulnerabilities not identified in this report.

Furthermore, **Fencer** shall not be held accountable for:

- Security vulnerabilities that may arise from changes to the systems after testing or that were not detectable during the test period.
- Despite diligent efforts to ensure the accuracy and completeness of the report's contents, the possibility of undetected flaws exists.
- Any loss resulting from critical information within the report being compromised due to loss or unauthorized acquisition by third parties.
- Implementation of the remediation recommendations within the report is deemed as effective solutions for the identified issues; however, should these implementations result in new issues within the institution's systems, it is advised to consult with support companies or specialist consultants in the field for their perspectives.

# 2 Overview

This document provides a comprehensive remediation plan for the penetration test conducted for Honey Corp. It contains detailed guidance on addressing all 22 vulnerabilities identified during the engagement that are still open.

Of the total vulnerabilities identified, 22 remain in an open state. This report provides technical details and remediation steps for each finding to assist the development team in addressing remaining issues.

# 3 Scope

The scope of this penetration test was defined to assess the security posture of the following application surfaces belonging to Honey Corp.

| | |
|---|---|
| **Primary target** | https://honeyapp.honey-corp.com/ |
| **Engagement objective** | Perform a full penetration test of honeyapp.honey-corp.com across the entire web and API surface plus any AI/LLM features: recon and enumeration first, then validate and bounded-exploit vulnerabilities across every class. Chain confirmed findings into attack paths, and deliver the consolidated report. |
| **Goal** | pentest |

## 3.1 In-Scope hosts

- `https://honeyapp.honey-corp.com/`

## 3.2 Out-of-Scope

The following activities and systems were explicitly excluded from this testing engagement:

- Social engineering attacks against personnel
- Physical security assessments
- Denial of service (DoS) attacks
- Production data manipulation or deletion

# 4 Methodology

The test was done according to penetration testing best practices. The flow from start to finish is listed below.

This penetration test was performed by an AI agent using automated tools and analysis throughout the engagement.

## 4.1 Pre Engagement

- Scoping
- Customer documentation
- Information discovery

## 4.2 Penetration Testing

- Tool assisted assessment
- Manual assessment
- Exploitation
- Risk analysis
- Reporting

## 4.3 Post Engagement

- Prioritized remediation*
- Best practice support*
- Retesting*

## 5 Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

| Severity | CVSS Score Range | Definition |
|---|---|---|
| **Critical** | 9.0-10.0 | Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately. |
| **High** | 7.0-8.9 | Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible. |
| **Medium** | 4.0-6.9 | Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved. |
| **Low** | 0.1-3.9 | Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window. |
| **Informational** | N/A | No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation. |

## 5.1 Risk Factors

Each finding is assigned two factors to measure its risk. Factors are measured on a scale of 1 (very low) through 5 (very high).

**Impact**

This indicates the finding's effect on technical and business operations. It covers aspects such as the confidentiality, integrity, and availability of data or systems, and financial or reputational loss.

**Likelihood**

This indicates the finding's potential for exploitation. It takes into account aspects such as skill level required of an attacker and relative ease of exploitation.

## 5.2 Severity Definitions

When our pentesters find vulnerabilities, they use the standard OWASP Risk Rating Methodology, and then classify them into one of the following risk levels, based on their business impact and likelihood:

```
Risk = Impact * Likelihood
```

**Critical:** Includes vulnerabilities that require immediate attention. Risk score of 25.

**High:** Impacts the security of your application/platform/hardware, including supported systems. Includes high probability vulnerabilities with a high business impact. Risk score range: 16 through 24.

**Medium:** Includes vulnerabilities that are: medium risk, medium impact; low risk, high impact; high risk, low impact. Risk score range: 5 through 15.

**Low:** Specifies common vulnerabilities with minimal impact. Risk score range: 2 through 4.

**Informational:** Notes vulnerabilities of minimal risk to your business. Risk score of 1.

# 6 Vulnerability Remediation Status

The following is a summary of all vulnerabilities identified during the assessment. For detailed technical information and remediation steps, please refer to the section below.

| Vulnerability | Severity | Status | Discovered | Last Updated |
|---|---|---|---|---|
| Mass assignment allows self-registration as admin | Critical | Open | 2026-08-03 | 2026-08-20 |
| Credential leak via /rest/memories exposes all password has… | Critical | Open | 2026-08-03 | 2026-08-20 |
| SQL injection in product search and login endpoints | Critical | Open | 2026-08-03 | 2026-08-20 |
| JWT algorithm bypass enables token forgery | Critical | Open | 2026-08-03 | 2026-08-20 |
| XXE via XML file upload enables arbitrary file read | High | Open | 2026-08-20 | 2026-08-20 |
| SSRF via profile image URL fetch | High | Open | 2026-08-20 | 2026-08-20 |
| Weak credential on admin account | High | Open | 2026-08-12 | 2026-08-20 |
| Unauthenticated PUT on Products enables price manipulation | High | Open | 2026-08-12 | 2026-08-20 |
| Password change without current password verification | High | Open | 2026-08-12 | 2026-08-20 |
| JWT payload contains password hashes | High | Open | 2026-08-03 | 2026-08-20 |
| DOM XSS via search parameter | High | Open | 2026-08-03 | 2026-08-20 |
| Encryption keys publicly accessible | High | Open | 2026-08-03 | 2026-08-20 |
| IDOR on user baskets allows cross-tenant data read | High | Open | 2026-08-03 | 2026-08-20 |
| Null-byte file extension bypass on /ftp/ | High | Open | 2026-08-03 | 2026-08-20 |
| NoSQL operator injection in product reviews | Medium | Open | 2026-08-20 | 2026-08-20 |
| Missing brute-force protection on login | Medium | Open | 2026-08-12 | 2026-08-20 |

| Vulnerability | Severity | Status | Discovered | Last Updated |
|---|---|---|---|---|
| LLM system prompt and tool code leaked | Medium | Open | 2026-08-03 | 2026-08-20 |
| Application configuration disclosure without authentication | Medium | Open | 2026-08-03 | 2026-08-20 |
| Exposed Swagger/OpenAPI documentation for B2B API | Low | Open | 2026-08-20 | 2026-08-20 |
| Wildcard CORS enables cross-origin data theft | Low | Open | 2026-08-12 | 2026-08-20 |
| Unauthenticated metrics exposure leaks operational intellig… | Low | Open | 2026-08-03 | 2026-08-20 |
| Open redirect via whitelist substring bypass | Low | Open | 2026-08-03 | 2026-08-20 |

# 7 Remediation Details

The following sections provide detailed remediation guidance for each currently unresolved vulnerability identified during the penetration test.

### Mass assignment allows self-registration as admin

Critical

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA6/

**Risk Description**

Any unauthenticated visitor can create an admin-level account, gaining full administrative access to the application including user management, configuration changes, and all protected resources.

**Reproduction Steps**

1. Send POST https://honeyapp.honey-corp.com/api/Users/ with body {"email":"test@example.com","password":"Password123!","role":"admin"}
2. Verify HTTP 201 response with role:admin in the created user object
3. Login with POST /rest/user/login using the new credentials
4. Verify the returned JWT contains role:admin in its payload

**Remediation**

1. Implement a server-side allowlist for user-settable fields during registration (email, password only)
2. Never allow the `role` field to be set via the public registration endpoint
3. Use a DTO/schema that explicitly excludes privileged fields from mass assignment
4. Add integration tests that verify role escalation via registration is rejected

### Credential leak via /rest/memories exposes all password hashes

Critical

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA8/

**Risk Description**

All user credentials (unsalted MD5 hashes) are publicly accessible without authentication. MD5 is trivially reversible via rainbow tables or hashcat, granting immediate access to all user accounts including admin.

**Proof of Impact**

Admin hash 0192023a7bbd73250516f069df18b500 cracked to 'admin123'. Login as admin@juice-sh.op confirmed.

**Reproduction Steps**

1. Send GET https://honeyapp.honey-corp.com/rest/memories without any authentication headers
2. Parse the JSON response and extract User objects from each memory entry
3. Identify admin users by role field (e.g., bjoern.kimminich@gmail.com, role:admin)
4. Crack the MD5 hash offline (e.g., 0192023a7bbd73250516f069df18b500 = admin123)
5. Login with POST /rest/user/login using the cracked credentials

**Remediation**

1. Remove User object expansion from the memories API response (return only userId reference)
2. Never expose password hashes in any API response regardless of authentication state
3. Migrate from unsalted MD5 to bcrypt/argon2 with per-user salt
4. Require authentication for the memories endpoint

## SQL injection in product search and login endpoints

Critical

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA3/

**Risk Description**

An unauthenticated attacker can:

- Extract the entire database including credit card numbers (PANs), user credentials, orders, and addresses via UNION injection
- Authenticate as ANY user without knowing their password via login SQLi
- Enumerate all database tables and schema

**Proof of Impact**

Extracted credit card: Bjoern Kimminich, PAN 481520XXXXXX2754, exp 12/2092. Authenticated as jim@juice-sh.op without password.

**Reproduction Steps**

1. Send GET https://honeyapp.honey-corp.com/rest/products/search?q=test%27))%20OR%201=1-- to confirm injection (returns all 56 products vs 0 for normal search)
2. Send GET /rest/products/search?q=qwert%27))%20UNION%20SELECT%20id,email,password,role,5,6,7,8,9%20FROM%20Users%20LIMIT%201--%20x to extract admin credentials
3. Send POST /rest/user/login with body {"email":"admin@juice-sh.op'--","password":"x"} to bypass authentication as admin
4. Verify returned JWT grants admin access via GET /api/Users/

**Remediation**

1. Replace all raw SQL string concatenation with parameterized queries/prepared statements
2. Use an ORM's query builder with parameter binding for all user-controlled inputs
3. Implement input validation rejecting SQL metacharacters in email and search fields
4. Apply least-privilege database permissions (read-only for search queries)

## JWT algorithm bypass enables token forgery

Critical

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA4/

**Risk Description**

Any unauthenticated attacker can forge admin-level JWT tokens without any secret or key material, gaining immediate full access to all authenticated and admin-only API endpoints.

**Proof of Impact**

Forged admin token listed all 41 users. Read basket/2 (Jim: 2x Raspberry Juice) and basket/3 (Bender: 1x Raspberry Juice).

**Reproduction Steps**

1. Base64url-encode header {"typ":"JWT","alg":"none"} and payload {"data":{"id":1,"email":"admin@juice-sh.op","role":"admin"},"iat":1724000000}
2. Concatenate as header.payload. (trailing dot, empty signature)
3. Send GET https://honeyapp.honey-corp.com/api/Users/ with Authorization: Bearer
4. Verify HTTP 200 with full user list returned (41 accounts)
5. Send GET /rest/basket/2 with same token to read another user's basket

**Remediation**

1. Reject JWTs with `alg: none` — enforce a whitelist of allowed algorithms (RS256 only)
2. Use a JWT library that refuses unsigned tokens by default
3. Validate the algorithm against a server-side configuration, never trust the token's declared algorithm
4. Consider adding token binding or audience validation

## XXE via XML file upload enables arbitrary file read

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BYZ/

**Risk Description**

An attacker can read any file accessible to the Node.js process on the server, including application source code, configuration files, /etc/passwd, environment variables, and cryptographic keys.

**Proof of Impact**

Read /etc/hostname: b9e08a63c357

**Reproduction Steps**

1. Create XML file: ]>&xxe;
2. Upload via POST /file-upload as multipart form-data with filename ending in .xml
3. Observe the entity is expanded in the error response body
4. The file contents (b9e08a63c357) appear where the entity reference was

**Remediation**

1. Disable external entity processing in libxmljs: set `noent: false, dtdload: false, dtdvalid: false`
2. If XML parsing is not required, reject XML uploads entirely
3. Use a safe XML parser configuration that strips DOCTYPE declarations
4. Implement file content validation before XML parsing

## SSRF via profile image URL fetch

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BYY/

**Risk Description**

An authenticated attacker can make the server fetch arbitrary internal URLs, accessing internal services, admin endpoints, and potentially cloud metadata services that are not externally accessible.

**Reproduction Steps**

1. Authenticate (or use forged JWT)
2. Send POST /profile/image/url with body imageUrl=http://localhost:3000/rest/admin/application-version
3. Retrieve the stored image at /assets/public/images/uploads/{userId}.jpg
4. Observe the internal service response is stored as the image content

**Remediation**

1. Implement a URL allowlist for the profile image fetch (only allow external HTTPS image URLs)
2. Block requests to private IP ranges (127.0.0.0/8, 10.0.0.0/8, 169.254.0.0/16, 172.16.0.0/12, 192.168.0.0/16)
3. Validate that the fetched content is actually an image (check Content-Type and magic bytes)
4. Use a dedicated image proxy service with network isolation

### Weak credential on admin account

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BNT/

**Risk Description**

The admin account uses a factory-default password trivially guessable by any attacker. Combined with the absence of rate limiting, this is recoverable in seconds.

**Reproduction Steps**

1. Send POST /rest/user/login with body {"email":"admin@juice-sh.op","password":"admin123"}
2. Verify HTTP 200 with authentication token returned
3. Decode the JWT to confirm role:admin

**Remediation**

1. Change the admin password immediately to a strong, unique value
2. Implement a password complexity policy (minimum length, character requirements)
3. Force password change on first login for default accounts
4. Consider disabling default accounts entirely in production

### Unauthenticated PUT on Products enables price manipulation

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BNW/

**Risk Description**

An unauthenticated attacker can manipulate product prices for financial fraud and inject stored XSS payloads into product descriptions that execute for all site visitors.

**Reproduction Steps**

1. Send PUT https://honeyapp.honey-corp.com/api/Products/1 with body {"price":0.01} (no auth headers)
2. Verify HTTP 200 with the modified price in the response
3. Confirm the price change persists by GET /api/Products/1

**Remediation**

1. Require admin authentication for all PUT/PATCH/DELETE operations on Products
2. Implement role-based access control at the API layer
3. Add request signing or CSRF protection for state-changing operations
4. Validate that price values are within acceptable business ranges

### Password change without current password verification

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BNV/

**Risk Description**

An attacker with any session token (stolen via XSS, intercepted, or forged) can permanently change a user's password without knowing the current one, locking the legitimate user out.

**Reproduction Steps**

1. Obtain any valid JWT (via XSS theft, alg:none forgery, or login)
2. Send GET /rest/user/change-password?new=NewPassword1&repeat=NewPassword1 with the JWT
3. Verify HTTP 200 — password changed without current password validation
4. Note: the original JWT remains valid even after password change

**Remediation**

1. Require the current password field and validate it before allowing a change
2. Use POST method instead of GET for state-changing operations
3. Invalidate all existing tokens/sessions when password is changed
4. Implement re-authentication for sensitive account operations

### JWT payload contains password hashes

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAA/

**Risk Description**

Password hashes are transmitted to and stored on every client that authenticates. Combined with the confirmed DOM XSS and missing HttpOnly cookie flag, any XSS vector can exfiltrate password hashes for offline cracking.

**Reproduction Steps**

1. Login via POST /rest/user/login with valid credentials
2. Decode the returned JWT (base64url decode the payload section)
3. Observe the password field contains the MD5 hash of the user's password
4. Crack the hash offline (0192023a7bbd73250516f069df18b500 = admin123)

**Remediation**

1. Remove sensitive fields (password, totpSecret, deluxeToken) from the JWT payload
2. Include only the minimum claims needed for authorization (id, email, role)
3. Migrate to opaque session tokens that reference server-side state
4. Set HttpOnly and Secure flags on all authentication cookies

## DOM XSS via search parameter

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA5/

**Risk Description**

An attacker can craft a malicious link that, when clicked by a victim, executes arbitrary JavaScript in the victim's browser session. This enables session theft (JWT from localStorage), account takeover, and data exfiltration.

**Proof of Impact**

JavaScript execution confirmed via alert('fencerXSS8421') in application origin.

**Reproduction Steps**

1. Navigate to https://honeyapp.honey-corp.com/#/search?q=%3Ciframe%20src%3D%22javascript%3Aalert('XSS')%22%3E
2. Observe JavaScript alert dialog fires in the browser
3. Verify execution is in the application's origin (document.domain = honeyapp.honey-corp.com)

**Remediation**

1. Remove the `bypassSecurityTrustHtml()` call on user-controlled search input
2. Use Angular's default sanitization for the search value display
3. Deploy a Content-Security-Policy header that blocks inline scripts and unsafe-eval
4. Consider using textContent instead of innerHTML for displaying search terms

## Encryption keys publicly accessible

High

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAF/

**Risk Description**

Exposed JWT public key enables algorithm confusion attacks. The premium.key is a live shared secret that must be rotated immediately.

**Reproduction Steps**

1. Browse GET https://honeyapp.honey-corp.com/encryptionkeys/ to see directory listing
2. Download GET /encryptionkeys/jwt.pub to obtain the RSA public key
3. Download GET /encryptionkeys/premium.key to obtain the shared secret
4. Use the public key for HS256 algorithm confusion JWT forgery

**Remediation**

1. Remove the /encryptionkeys/ directory from public web access
2. Store cryptographic keys in environment variables or a secrets manager
3. Rotate the premium.key immediately as it is compromised
4. Ensure JWT verification strictly validates the expected algorithm (RS256 only)

## IDOR on user baskets allows cross-tenant data read

High

### Fencer Vulnerability Details

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA7/

### Risk Description

An attacker can read any user's shopping basket contents, revealing their purchase intentions, quantities, and product selections — private commercial data that should be isolated per user.

### Proof of Impact

Read Jim's basket (id:2): 2x Raspberry Juice. Read Bender's basket (id:3): 1x Raspberry Juice.

### Reproduction Steps

1. Obtain any valid JWT (via login, registration, or alg:none forgery)
2. Send GET https://honeyapp.honey-corp.com/rest/basket/2 with the JWT in Authorization header
3. Verify HTTP 200 with another user's basket contents (UserId:2 = jim@juice-sh.op)
4. Iterate basket IDs (1-5) to read multiple users' baskets

### Remediation

1. Enforce ownership checks: basket endpoint must verify the requesting user's ID matches the basket's UserId
2. Use the authenticated user's ID from the JWT to look up THEIR basket, not accept arbitrary IDs
3. Return 403 when the basket ID does not belong to the requesting user

## Null-byte file extension bypass on /ftp/

High

### Fencer Vulnerability Details

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BA9/

### Risk Description

An attacker can download any file in the /ftp/ directory regardless of extension restrictions, including backup files with credentials, KeePass databases, compiled Python code, and application configuration.

### Reproduction Steps

1. Browse GET https://honeyapp.honey-corp.com/ftp/ to see directory listing
2. Attempt GET /ftp/package.json.bak and observe 403 'Only .md and .pdf' error
3. Send GET /ftp/package.json.bak%2500.md (URL-encoded %00 null byte + .md extension)
4. Verify HTTP 200 with full file contents returned

### Remediation

1. Remove the /ftp/ directory from public web access entirely
2. If file serving is needed, use a proper allowlist based on actual file content type, not extension string matching
3. Sanitize file path inputs to reject null bytes and path traversal characters
4. Move sensitive files to a non-web-accessible storage location

## NoSQL operator injection in product reviews

Medium

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1C0E/

**Risk Description**

An attacker can modify ALL product reviews simultaneously by injecting MongoDB operators, enabling mass defacement, spam injection, or reputation manipulation across the entire product catalog.

**Reproduction Steps**

1. Send PATCH https://honeyapp.honey-corp.com/rest/products/reviews with body {"id":{"$ne":""},"message":"test"}
2. Observe response shows modified:31 (all reviews matched)
3. Verify via GET /rest/products/1/reviews that review messages were changed

**Remediation**

1. Validate that the `id` field is a string/ObjectId, rejecting objects and operators
2. Use MongoDB's strict query mode or schema validation
3. Implement input type checking before passing to MongoDB queries
4. Add authentication requirement for PATCH operations on reviews

## Missing brute-force protection on login

Medium

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BNS/

**Risk Description**

Unrestricted login attempts enable credential spraying and brute-force attacks at full speed. Combined with leaked user emails, any weak password in the user base is recoverable.

**Reproduction Steps**

1. Run login_throttle_probe against POST /rest/user/login with a test email
2. Observe 25 consecutive failed attempts all return HTTP 401 with no delay or blocking
3. Note absence of: rate limit headers, 429 responses, CAPTCHA challenges, account lockout

**Remediation**

1. Implement progressive rate limiting (e.g., exponential backoff after 5 failures)
2. Add account lockout after 10 consecutive failures (with admin unlock)
3. Deploy CAPTCHA after 3 failed attempts
4. Consider IP-based rate limiting at the application or WAF layer

### LLM system prompt and tool code leaked

Medium

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAD/

**Risk Description**

Leaked system prompt reveals hidden business logic (confidential discount policy) that can be exploited via prompt injection. Tool source code exposes a NoSQL injection vector and internal infrastructure details.

**Reproduction Steps**

1. Send GET https://honeyapp.honey-corp.com/snippets/chatbotGreedyInjectionChallenge
2. Read the full system prompt including CONFIDENTIAL sections
3. Note the 15% discount policy and $where NoSQL injection pattern

**Remediation**

1. Remove or restrict access to code snippet endpoints
2. Do not embed confidential business policies in system prompts
3. Fix the NoSQL $where injection in the getProductReviews tool
4. Use parameterized queries instead of string concatenation in MongoDB operations

### Application configuration disclosure without authentication

Medium

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAB/

**Risk Description**

Leaked security question answers enable direct password reset for specific accounts. OAuth credentials and internal configuration expose the application's authentication infrastructure.

**Reproduction Steps**

1. Send GET https://honeyapp.honey-corp.com/rest/admin/application-configuration without any auth
2. Parse the JSON response for sensitive configuration data
3. Note the security question answers and OAuth client ID

**Remediation**

1. Require admin authentication for this endpoint
2. Remove security question answers from the configuration response
3. Move OAuth secrets to environment variables not exposed via any API
4. Implement proper access control on all /rest/admin/* endpoints

### Exposed Swagger/OpenAPI documentation for B2B API

Low

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1C0J/

**Risk Description**

Full API contract disclosed to unauthenticated users. Provides attackers with complete endpoint inventory, parameter types, and authentication requirements for targeted abuse.

**Reproduction Steps**

1. Browse https://honeyapp.honey-corp.com/api-docs/
2. View the full B2B API specification including endpoints, schemas, and auth requirements

**Remediation**

1. Require authentication to access API documentation
2. Disable Swagger UI in production environments
3. If documentation must be public, ensure it does not expose internal-only endpoints

### Wildcard CORS enables cross-origin data theft

Low

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BNX/

**Risk Description**

Any malicious website can read the application's unauthenticated API responses cross-origin, including PII (user emails in reviews/feedback) and sensitive configuration data.

**Reproduction Steps**

1. Send any API request with Origin: https://evil.com header
2. Observe Access-Control-Allow-Origin: * in response
3. Note that all unauthenticated endpoint data is readable cross-origin

**Remediation**

1. Replace wildcard CORS with an explicit allowlist of trusted origins
2. For endpoints requiring credentials, ensure Access-Control-Allow-Credentials is not combined with wildcard origin
3. Restrict allowed methods to only those needed per endpoint

### Unauthenticated metrics exposure leaks operational intelligence

Low

**Fencer Vulnerability Details**

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAG/

**Risk Description**

Operational metrics reveal exact software versions for CVE targeting, user counts for attack planning, and business metrics that should remain internal.

**Reproduction Steps**

1. Send GET https://honeyapp.honey-corp.com/metrics without authentication
2. Parse Prometheus text format for operational data

**Remediation**

1. Require authentication for the /metrics endpoint
2. Restrict access to internal monitoring infrastructure only
3. Deploy a reverse proxy rule blocking external access to /metrics

## Open redirect via whitelist substring bypass

Low

### Fencer Vulnerability Details

https://app.fencer.dev/a/honey/vulnerabilities/VULN-1BAC/

### Risk Description

An attacker can craft trusted-looking URLs that redirect victims to phishing or malware sites, abusing the honeyapp.honey-corp.com domain's trust.

### Reproduction Steps

1. Send GET /redirect?to=https://evil.com%23https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm
2. Observe HTTP 302 redirect to evil.com (whitelisted string in fragment passes check)

### Remediation

1. Parse the URL and validate only the scheme+host portion against the allowlist
2. Do not use substring matching for URL validation
3. Reject URLs containing fragments or unexpected characters in the host portion

## 8 Engagement Artifacts

State the engagement created or changed on the target is recorded below, including whether it was reverted and any cleanup still required.

| Type | Identifier | Created via | Reverted | Cleanup |
|---|---|---|---|---|
| user_account | pentest-probe@test.invalid (id:38, role:admin) `https://honeyapp.honey-corp.com/api/Users/38` | POST /api/Users/ with role:admin (mass assignment test) | No | Yes Delete u probe@ table via databas |
| user_account | authtest-admin1@test.com (id:37, role:admin) `https://honeyapp.honey-corp.com/api/Users/37` | POST /api/Users/ with role:admin (auth testing) | No | Yes Delete u admin1( table via databas |
| user_account | fencer-test@juice-sh.op (id:36) `https://honeyapp.honey-corp.com/api/Users/36` | POST /api/Users/ (server-side testing account for SSRF) | No | Yes Delete u test@jui table. |
| uploaded_file | /assets/public/images/uploads/36.jpg (contains SSRF response) `https://honeyapp.honey-corp.com/assets/public/images/uploads/36.jpg` | POST /profile/image/url with imageUrl=http://localhost:3000/rest/admin/application-version | No | Yes Delete th /assets/ from the director |
| config_change | Product 1 description changed to 'test' (was 'The all-time classic.') `https://honeyapp.honey-corp.com/api/Products/1` | PUT /api/Products/1 {"description":"test"} (unauthenticated PUT test) | No | Yes Restore {"descri or direct |
| config_change | 31 product reviews modified (message field set to 'nosql-probe') `https://honeyapp.honey-corp.com/rest/products/reviews` | PATCH /rest/products/reviews with {"id":{"$ne":""},"message":"nosql-probe"} | No | Yes Restore databas affected reviews |

# 9 Recommendations

## 9.1 Immediate Actions Required

- **Critical Vulnerabilities:** Address all 4 critical vulnerabilities immediately. These pose significant risk to the organization.
- **High Severity Vulnerabilities:** Remediate 10 high severity vulnerabilities within 30 days.

## 9.2 General Security Recommendations

- Implement a regular vulnerability scanning schedule
- Establish a vulnerability management process
- Provide security training to development teams
- Implement secure coding practices
- Regular security assessments and penetration testing

# 10 Appendix

## 10.1 Test Summary

| Metric | Count |
| --- | --- |
| Total Vulnerabilities | 22 |
| Critical Severity | 4 |
| High Severity | 10 |
| Medium Severity | 4 |
| Low Severity | 4 |
| Informational | 0 |