SECURITY ANALYSIS REPORT

Generated by Backstage Rookie

Provider: Backstage Rookie

contact - swarajdarekar9@gmail.com

Client: heaven Darekar

darekarheaven@gmail.com

Scan ID: f72882f4-802f-4518-855f-0513f4c0a125

Date: 2026-02-06 18:01:34

Version: 1.0.0

CONFIDENTIAL

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report summarizes the security posture of the application based on a comprehensive static analysis. Key findings include critical vulnerabilities related to sensitive information exposure and insecure operational configurations, as well as multiple instances of missing safeguards in external communications and verbose error logging. These issues collectively present a significant risk, ranging from potential unauthorized access and denial-of-service to information leakage, which could be exploited by malicious actors.

Identified Vulnerabilities

ID	Title	CVSS	Page
C1	Hardcoded JWT Token Exposure	9.8	tasks.py:14
C2	Flask Debug Mode Enabled in Production	9.8	app.py:1872
M1	Missing Request Timeout for External API Calls	7.5	routes\payments.py:90
M2	Missing Request Timeout for External API Calls	7.5	routes\payments.py:136
M3	Missing Request Timeout for External API Calls	7.5	routes\payments.py:663
L1	Verbose Error Messages Exposing Internal Details	3.0	app.py:117

METHODOLOGY

Introduction

This report details the results of a security assessment conducted on the specified repository. The analysis involved a multi-layered approach, combining automated static analysis tools with advanced, AI-driven verification and enrichment to identify potential security vulnerabilities.

Objective

The primary objective of this assessment was to identify security weaknesses, assess their potential impact, and provide actionable recommendations for remediation to improve the overall security posture of the application.

Scope

The assessment was performed on the source code of the repository cloned at the time of the scan. The analysis focused on common web application vulnerabilities, insecure coding practices, and dependency risks.

Systems in Scope

No systems explicitly defined.

User Accounts

As this was a static source code analysis, no user accounts were provisioned or tested.

FINDINGS

C1 – Hardcoded JWT Token Exposure

Severity:	Critical
CVSS Score:	9.8
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Target:	Application codebase and exposed data

Overview

A full JSON Web Token (JWT) is hardcoded directly within the `tasks.py` file. While this might be intended for testing or demonstration purposes, the presence of a hardcoded, valid JWT in the codebase poses a significant security risk.

Details

The `tasks.py` file contains a line (line 14) where a complete JWT token, with an embedded `exp` (expiration) timestamp indicating validity until late 2025, is assigned to a variable. This token includes sensitive information such as issuer (`iss`), subject (`sub`), audience (`aud`), email, phone, role, and other user metadata. If this token was ever used in a production or staging environment, or if it provides a template that attackers could use to craft valid-looking tokens, it could lead to unauthorized access, impersonation, or sensitive data disclosure. Even if intended for local testing, its exposure in a public or internal repository could reveal internal system structure, user roles, and email formats. The presence of the Supabase URL as the issuer further ties this to a specific service.

Evidence

- **tasks.py:14:** token = "eyJhbGciOiJIUzI1NiIsImtpZCI6IkRUK3NRQURhd0Uwdm9RbkwiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL2Frd2VtZWZld2ZtenhlZ2lkZXNsLnN1cGFiYXNlLmNvL2F1dGgvdjEi

References

- https://cheatsheetseries.owasp.org/cheatsheets/Hardcoded_secrets_Cheat_Sheet.html
- https://jwt.io/introduction

Recommendation

- Remove hardcoded sensitive tokens from the codebase immediately. - Store JWTs or other sensitive data used for testing in secure, version-controlled configuration files (e.g., `.env` files, `.test_secrets`) that are excluded from production deployments and public repositories. - If this token was valid in any environment, invalidate it and any associated user sessions. - Ensure that development and testing artifacts containing sensitive data are never included in production builds or accessible via public-facing repositories.

C2 – Flask Debug Mode Enabled in Production

Severity:	Critical
CVSS Score:	9.8
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Target:	Web application server

Overview

The Flask application is configured to run with debug mode enabled (`debug=True`) in a potentially production-facing context. This setting exposes the Werkzeug debugger, which is a critical security risk.

Details

When `debug=True` is set, the Flask application provides detailed error messages and an interactive debugger in the browser upon encountering an unhandled exception. While invaluable for development, this debugger allows an attacker with access to the debug PIN (which can sometimes be guessed or leaked) to execute arbitrary Python code on the server. This can lead to full system compromise, data exfiltration, or denial of service. The code snippet `if __name__ == '__main__': app.run(debug=True, ...)` suggests this configuration might be used when running the application directly, which could happen in development, but should be prevented in production deployment mechanisms (e.g., Gunicorn, uWSGI).

Evidence

- **app.py:1872:** app.run(debug=True, port=5000, use_reloader=True)

References

- https://flask.palletsprojects.com/en/latest/quickstart/#debug-mode
- https://cheatsheetseries.owasp.org/cheatsheets/Python_Flask_Cheat_Sheet.html#production-deployment

Recommendation

- Disable Flask debug mode (`debug=False`) for all production deployments. - Use a production-ready WSGI server (e.g., Gunicorn, uWSGI, Nginx/Apache with mod_wsgi) that does not directly invoke `app.run()`. - Implement robust error logging and monitoring instead of relying on the interactive debugger for production issues. - Ensure that sensitive configuration variables, including debug settings, are properly managed using environment variables or a secure configuration management system.

M1 – Missing Request Timeout for External API Calls

Severity:	Medium
CVSS Score:	7.5
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Target:	External API integrations

Overview

Multiple HTTP requests made to external services (e.g., Cashfree API) using the `requests` library do not specify a timeout. This can lead to the application hanging indefinitely if the external service is slow or unresponsive.

Details

Without a defined `timeout` parameter, the `requests` library will wait indefinitely for a response from the server. If an external API (like Cashfree's payment gateway) experiences delays or becomes unavailable, the application's threads or processes making these requests could become blocked, leading to resource exhaustion, application unresponsiveness, and potential denial-of-service (DoS) for legitimate users. This impacts the reliability and resilience of the payment processing logic.

Evidence

- **routes\payments.py:90:** response = requests.post(f"{CASHFREE_BASE_URL}/orders", headers=get_cashfree_headers(), json=payload)

References

- https://docs.python-requests.org/en/latest/user/advanced/#timeouts
- https://owasp.org/www-community/attacks/Denial_of_Service

Recommendation

- Add a `timeout` parameter to all `requests` calls to external services. - Choose appropriate timeout values based on the expected response times of the external APIs and the application's availability requirements. A tuple `(connect_timeout, read_timeout)` is often recommended for granular control. - Implement retry mechanisms with exponential backoff for transient network errors. - For example: `requests.post(url, json=payload, headers=headers, timeout=(3, 30))` where 3 seconds is the connect timeout and 30 seconds is the read timeout.

M2 – Missing Request Timeout for External API Calls

Severity:	Medium
CVSS Score:	7.5
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Target:	External API integrations

Overview

Multiple HTTP requests made to external services (e.g., Cashfree API) using the `requests` library do not specify a timeout. This can lead to the application hanging indefinitely if the external service is slow or unresponsive.

Details

Without a defined `timeout` parameter, the `requests` library will wait indefinitely for a response from the server. If an external API (like Cashfree's payment gateway) experiences delays or becomes unavailable, the application's threads or processes making these requests could become blocked, leading to resource exhaustion, application unresponsiveness, and potential denial-of-service (DoS) for legitimate users. This impacts the reliability and resilience of the payment processing logic.

Evidence

- **routes\payments.py:136:** response = requests.get(f"{CASHFREE_BASE_URL}/orders/{order_id}", headers=get_cashfree_headers())

References

- https://docs.python-requests.org/en/latest/user/advanced/#timeouts
- https://owasp.org/www-community/attacks/Denial_of_Service

Recommendation

- Add a `timeout` parameter to all `requests` calls to external services. - Choose appropriate timeout values based on the expected response times of the external APIs and the application's availability requirements. A tuple `(connect_timeout, read_timeout)` is often recommended for granular control. - Implement retry mechanisms with exponential backoff for transient network errors. - For example: `requests.post(url, json=payload, headers=headers, timeout=(3, 30))` where 3 seconds is the connect timeout and 30 seconds is the read timeout.

M3 – Missing Request Timeout for External API Calls

Severity:	Medium
CVSS Score:	7.5
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Target:	External API integrations

Overview

Multiple HTTP requests made to external services (e.g., Cashfree API) using the `requests` library do not specify a timeout. This can lead to the application hanging indefinitely if the external service is slow or unresponsive.

Details

Without a defined `timeout` parameter, the `requests` library will wait indefinitely for a response from the server. If an external API (like Cashfree's payment gateway) experiences delays or becomes unavailable, the application's threads or processes making these requests could become blocked, leading to resource exhaustion, application unresponsiveness, and potential denial-of-service (DoS) for legitimate users. This impacts the reliability and resilience of the payment processing logic.

Evidence

- **routes\payments.py:663:** payout_response = requests.post(payout_api_url, json=cashfree_payout_payload, headers=cashfree_headers)

References

- https://docs.python-requests.org/en/latest/user/advanced/#timeouts
- https://owasp.org/www-community/attacks/Denial_of_Service

Recommendation

- Add a `timeout` parameter to all `requests` calls to external services. - Choose appropriate timeout values based on the expected response times of the external APIs and the application's availability requirements. A tuple `(connect_timeout, read_timeout)` is often recommended for granular control. - Implement retry mechanisms with exponential backoff for transient network errors. - For example: `requests.post(url, json=payload, headers=headers, timeout=(3, 30))` where 3 seconds is the connect timeout and 30 seconds is the read timeout.

L1 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:117:** return jsonify({'msg': 'Registration failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L2 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:187:** return jsonify({'msg': 'Login failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L3 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:210:** return jsonify({'msg': 'Failed to update profile', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L4 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:243:** return jsonify({'msg': 'Failed to send reset email', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L5 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:288:** return jsonify({'msg': 'Failed to create campaign', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L6 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:326:** return jsonify({'msg': 'Failed to fetch campaigns', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L7 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:385:** return jsonify({'msg': 'Failed to fetch campaigns', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L8 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:491:** return jsonify({'msg': 'Failed to fetch campaign details', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L9 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:579:** return jsonify({'msg': 'Failed to fetch creator campaigns', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L10 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:653:** return jsonify({'msg': 'Failed to submit clip', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L11 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:718:** return jsonify({'msg': 'Failed to fetch clips', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L12 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:754:** return jsonify({'msg': 'Failed to fetch accepted clip details', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L13 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:829:** return jsonify({'msg': 'Failed to delete campaign', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L14 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:884:** return jsonify({'msg': 'Failed to delete clip', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L15 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:931:** return jsonify({'msg': 'Failed to fetch campaigns', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L16 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1022:** return jsonify({'msg': 'Failed to update clip', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L17 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1075:** return jsonify({'msg': 'Failed to delete clip', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L18 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1109:** return jsonify({'msg': 'Failed to fetch creator profile', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L19 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1155:** return jsonify({'msg': 'Failed to update creator profile', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L20 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1194:** return jsonify({'msg': 'Failed to update campaign image', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L21 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1226:** return jsonify({'msg': 'Failed to update campaign budget', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L22 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1255:** return jsonify({'msg': 'Failed to update campaign requirements', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L23 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1286:** return jsonify({'msg': 'Failed to update campaign status', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L24 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1317:** return jsonify({'msg': 'Failed to update campaign view threshold', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L25 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1354:** return jsonify({'msg': 'Failed to update campaign deadline', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L26 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1450:** return jsonify({'msg': 'Failed to retrieve pending payouts', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L27 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1478:** return jsonify({'msg': 'Failed to fetch brand profile', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L28 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1509:** return jsonify({'msg': 'Failed to update brand profile', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L29 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1576:** return jsonify({'msg': 'Failed to update view count', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L30 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1631:** return jsonify({'msg': 'Failed to update campaign views', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L31 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1729:** return jsonify({'msg': 'Failed to retrieve analytics', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L32 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1756:** return jsonify({'msg': 'Failed to sync Google user: could not retrieve user data', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L33 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1808:** return jsonify({'msg': 'Sync failed due to database error', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L34 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **app.py:1812:** return jsonify({'msg': 'Sync failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L35 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:115:** return jsonify({'msg': 'Internal server error', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L36 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:186:** return jsonify({'msg': 'Verification failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L37 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:343:** return jsonify({'msg': 'Allocation failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L38 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:430:** return jsonify({'msg': 'Reclaim failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L39 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:554:** return jsonify({'msg': 'Distribution failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L40 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:729:** return jsonify({'msg': 'Withdrawal failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L41 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:817:** return jsonify({'msg': 'Failed to save payout details', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L42 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:867:** return jsonify({'msg': 'Failed to retrieve payout details', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L43 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:927:** return jsonify({'msg': 'Failed to verify payout details', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L44 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:989:** return jsonify({'msg': 'Failed to retrieve withdrawal history', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L45 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1023:** return jsonify({'msg': 'Failed to retrieve notifications', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L46 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1105:** return jsonify({'msg': 'Failed to retrieve transactions', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L47 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1197:** return jsonify({'msg': 'Refund failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L48 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1264:** return jsonify({'msg': 'Failed to retrieve campaign summary', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L49 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1358:** return jsonify({'msg': 'Failed to calculate earnings', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L50 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1529:** return jsonify({'msg': 'Bulk distribution failed', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L51 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1623:** return jsonify({'msg': 'Failed to request refund', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L52 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1690:** return jsonify({'msg': 'Failed to retrieve refund requests', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L53 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1804:** return jsonify({'msg': 'Failed to approve refund', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L54 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1865:** return jsonify({'msg': 'Failed to reject refund', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L55 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:1934:** return jsonify({'msg': 'Failed to retrieve refund status', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L56 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:2014:** return jsonify({'msg': 'Failed to retrieve refund audit trail', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

L57 – Verbose Error Messages Exposing Internal Details

Severity:	Low
CVSS Score:	3.0
CVSS Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Target:	API endpoints

Overview

The application's API endpoints frequently return raw Python exception details (`str(e)`) in their error responses, which can expose internal system information to attackers.

Details

When an unexpected error occurs, the application captures the exception object and includes its string representation directly in the JSON response sent to the client. This verbose error messaging can inadvertently leak sensitive information such as file paths, database query details, internal library versions, environment variables, or other architectural insights. Attackers can leverage this information to map the application's attack surface, identify potential weak points, or gather intelligence for more targeted attacks. While beneficial for debugging during development, this practice significantly increases risk in production environments.

Evidence

- **routes\payments.py:2088:** return jsonify({'msg': 'Failed to revert withdrawal', 'error': str(e)}), 500

References

- https://owasp.org/www-community/Improper_Error_Handling

Recommendation

- Implement generic error messages for production environments. Do not return raw exception details to the client. - Log detailed exception information internally on the server for debugging purposes (e.g., to a secure log aggregation service). - For client-facing error messages, provide only high-level, user-friendly information (e.g., "An unexpected error occurred. Please try again later."). - Consider using custom exception handling middleware or decorators to standardize error responses and prevent accidental leakage of sensitive data.

ENDPOINT SECURITY ANALYSIS

This section provides a detailed security-oriented analysis of the identified API endpoints, including authentication mechanisms, data handling characteristics, potential security risks, and regulatory compliance considerations.

Endpoint Path /register

Endpoint Path	/register
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 73
Authentication Required	No
Risk Severity	Critical
CVSS Score	10.0

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (high): Authentication is missing for the registration endpoint.

  Potential Attack Scenario:

  An unauthenticated user could create multiple accounts, potentially for spam or abuse.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	high	Lack of authentication on user registration can lead to unauthorized account creation, violating principles of access control and integrity.
ISO/IEC 27001	Yes	high	Inadequate access control measures on user registration can undermine the confidentiality and integrity of user data.
CSA STAR	Yes	high	Absence of proper authentication for user registration increases the risk of unauthorized access and potential abuse of the system.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

References

- https://nvd.nist.gov/vuln/detail/CVE-2021-44228
- https://owasp.org/www-community/vulnerabilities/Authentication_and_authorization

Endpoint Path /login

Endpoint Path	/login
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 120
Authentication Required	No
Risk Severity	Critical
CVSS Score	10.0

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (high): Authentication is missing for the login endpoint.

  Potential Attack Scenario:

  An unauthenticated user could attempt to log in with arbitrary credentials, potentially leading to brute-force attacks or credential stuffing if rate limiting or other protections are absent.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	high	Lack of authentication controls on the login endpoint can lead to unauthorized access attempts, compromising the security of user accounts.
ISO/IEC 27001	Yes	high	Weak or missing authentication mechanisms on login endpoints increase the risk of unauthorized access and data breaches.
CSA STAR	Yes	high	Absence of proper authentication for the login process exposes the system to brute-force attacks and unauthorized access.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

References

- https://owasp.org/www-project-top-ten/2017/A1_-_Broken_Access_Control
- https://owasp.org/www-community/vulnerabilities/Brute_force_attacks

Endpoint Path /verify-instagram

Endpoint Path	/verify-instagram
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 192
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 191, 191, 192, 192, 193, 193, 194, 194, 191, 191, 193, 193, 194, 194, 191, 191, 193, 193, 194, 194.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	The endpoint has token-based authentication, which is a standard security measure for API access.
ISO/IEC 27001	Yes	low	Token-based authentication contributes to access control, aligning with ISO 27001 requirements.
CSA STAR	Yes	low	Use of tokens for authentication is a common practice that enhances security in cloud environments.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /verify-instagram/

Endpoint Path	/verify-instagram/
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 192
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 191, 191, 192, 192, 193, 193, 194, 194, 191, 191, 193, 193, 194, 194, 191, 191, 193, 193, 194, 194.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	The endpoint has token-based authentication, which is a standard security measure for API access.
ISO/IEC 27001	Yes	low	Token-based authentication contributes to access control, aligning with ISO 27001 requirements.
CSA STAR	Yes	low	Use of tokens for authentication is a common practice that enhances security in cloud environments.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /request-password-reset

Endpoint Path	/request-password-reset
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 224
Authentication Required	No
Risk Severity	High
CVSS Score	7.5

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (medium): Authentication is missing for the password reset request endpoint.

  Potential Attack Scenario:

  An unauthenticated attacker could trigger numerous password reset requests for users, potentially leading to denial-of-service or social engineering attacks.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	medium	Lack of authentication for password reset requests can be exploited for denial-of-service or account takeover attempts.
ISO/IEC 27001	Yes	medium	Inadequate controls around password reset mechanisms can lead to unauthorized access or disruption of service.
CSA STAR	Yes	medium	Unauthenticated password reset requests pose a risk of abuse, impacting the availability and integrity of user accounts.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

References

- https://owasp.org/www-community/vulnerabilities/Password_reset_vulnerabilities

Endpoint Path /api/brand/campaigns

Endpoint Path	/api/brand/campaigns
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 248
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 247, 247, 248, 248, 249, 249, 253, 253, 247, 247, 249, 249, 253, 253, 247, 247, 249, 249, 253, 253.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	The endpoint uses token authentication, which is appropriate for creating new resources.
ISO/IEC 27001	Yes	low	Authenticated access for campaign creation supports access control and integrity requirements.
CSA STAR	Yes	low	Token-based authentication for campaign creation is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns

Endpoint Path	/api/brand/campaigns
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 292
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 291, 291, 292, 292, 293, 293, 297, 297, 291, 291, 293, 293, 297, 297, 291, 291, 293, 293, 297, 297.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access for listing campaigns ensures that only authorized users can view this data.
ISO/IEC 27001	Yes	low	Token-based authentication supports access control for retrieving campaign data.
CSA STAR	Yes	low	Securing campaign listing with tokens is a standard practice for data access control.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/campaigns

Endpoint Path	/api/campaigns
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 329
Authentication Required	No
Risk Severity	Low
CVSS Score	2.7

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (low): Authentication is missing for the public campaigns endpoint.

  Potential Attack Scenario:

  While likely intended for public access, if sensitive campaign data is exposed, this could lead to information leakage.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Assuming this endpoint is intended for public access, lack of authentication is acceptable. However, data exposure must be carefully considered.
ISO/IEC 27001	Yes	low	Publicly accessible endpoints are acceptable if they do not expose sensitive information. Access controls should be reviewed for other endpoints.
CSA STAR	Yes	low	Publicly accessible data is common, but ensures no sensitive information is inadvertently exposed.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/campaigns/

Endpoint Path	/api/campaigns/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 388
Authentication Required	No
Risk Severity	Low
CVSS Score	2.7

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (low): Authentication is missing for retrieving a specific campaign by ID.

  Potential Attack Scenario:

  If campaign details are sensitive, this could lead to unauthorized information disclosure.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Assuming campaign details are public or accessible to all authenticated users, lack of specific authentication here might be acceptable. Review data sensitivity.
ISO/IEC 27001	Yes	low	Access control to campaign details needs to be verified. If sensitive, authentication should be enforced.
CSA STAR	Yes	low	The exposure of campaign details without authentication needs to be assessed for data sensitivity.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/your-campaigns

Endpoint Path	/api/creator/your-campaigns
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 495
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 494, 494, 495, 495, 496, 496, 501, 501, 494, 494, 496, 496, 501, 501, 494, 494, 496, 496, 501, 501.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can only view their own campaigns.
ISO/IEC 27001	Yes	low	Token-based authentication helps enforce access control for creator-specific data.
CSA STAR	Yes	low	Securing creator campaign data with tokens is a standard security measure.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/submit-clip

Endpoint Path	/api/creator/submit-clip
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 583
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 582, 582, 583, 583, 584, 584, 590, 590, 582, 582, 584, 584, 590, 590, 582, 582, 584, 584, 590, 590.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized submission of clips.
ISO/IEC 27001	Yes	low	Token-based authentication ensures that only authorized creators can submit clips.
CSA STAR	Yes	low	Securing the clip submission process with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/campaign-clips

Endpoint Path	/api/creator/campaign-clips
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 657
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 656, 656, 657, 657, 658, 658, 662, 662, 656, 656, 658, 658, 662, 662, 656, 656, 658, 658, 662, 662.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can only view clips associated with their campaigns.
ISO/IEC 27001	Yes	low	Token-based authentication enforces access control for viewing creator-specific clip data.
CSA STAR	Yes	low	Securing creator clip data with tokens is a standard security measure.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/accepted-clip-details/

Endpoint Path	/api/creator/accepted-clip-details/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 722
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 721, 721, 722, 722, 723, 723, 727, 727, 721, 721, 723, 723, 727, 727, 721, 721, 723, 723, 727, 727.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can only view details of their accepted clips.
ISO/IEC 27001	Yes	low	Token-based authentication enforces access control for viewing clip details.
CSA STAR	Yes	low	Securing clip details with tokens is a standard security measure.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns/

Endpoint Path	/api/brand/campaigns/
HTTP Methods	DELETE
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 761
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 761, 761, 765, 765, 766, 766, 770, 770, 765, 765, 766, 766, 770, 770, 765, 765, 766, 766, 770, 770.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated DELETE operation ensures only authorized users can remove campaigns.
ISO/IEC 27001	Yes	low	Token-based authentication for campaign deletion supports access control and integrity.
CSA STAR	Yes	low	Securing campaign deletion with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/clip/

Endpoint Path	/api/creator/clip/
HTTP Methods	DELETE
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 833
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 832, 832, 833, 833, 837, 837, 838, 838, 842, 842, 832, 832, 837, 837, 838, 838, 842, 842, 832, 832, 837, 837, 838, 838, 842, 842.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated DELETE operation ensures only creators can delete their own clips.
ISO/IEC 27001	Yes	low	Token-based authentication for clip deletion supports access control and integrity.
CSA STAR	Yes	low	Securing clip deletion with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/campaigns

Endpoint Path	/api/admin/campaigns
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 888
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 887, 887, 888, 888, 889, 889, 887, 887, 889, 889, 887, 887, 889, 889.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access for administrators to view campaigns ensures proper oversight.
ISO/IEC 27001	Yes	low	Token-based authentication for admin access aligns with access control requirements.
CSA STAR	Yes	low	Securing administrative access to campaign data with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/clip/

Endpoint Path	/api/admin/clip/
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 935
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 934, 934, 935, 935, 936, 936, 934, 934, 936, 936, 934, 934, 936, 936.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures only administrators can update clip details.
ISO/IEC 27001	Yes	low	Token-based authentication for admin clip updates supports access control and integrity.
CSA STAR	Yes	low	Securing admin clip updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/clip/

Endpoint Path	/api/admin/clip/
HTTP Methods	DELETE
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1026
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1025, 1025, 1026, 1026, 1031, 1031, 1025, 1025, 1031, 1031, 1025, 1025, 1031, 1031.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated DELETE operation ensures only administrators can delete clips.
ISO/IEC 27001	Yes	low	Token-based authentication for admin clip deletion supports access control and integrity.
CSA STAR	Yes	low	Securing admin clip deletion with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/profile

Endpoint Path	/api/creator/profile
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1079
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1078, 1078, 1079, 1079, 1080, 1080, 1084, 1084, 1078, 1078, 1080, 1080, 1084, 1084, 1078, 1078, 1080, 1080, 1084, 1084.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can only view their own profile.
ISO/IEC 27001	Yes	low	Token-based authentication enforces access control for creator profile data.
CSA STAR	Yes	low	Securing creator profile data with tokens is a standard security measure.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/creator/profile

Endpoint Path	/api/creator/profile
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1113
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1112, 1112, 1113, 1113, 1114, 1114, 1118, 1118, 1112, 1112, 1114, 1114, 1118, 1118, 1112, 1112, 1114, 1114, 1118, 1118.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures creators can update only their own profile.
ISO/IEC 27001	Yes	low	Token-based authentication for creator profile updates supports access control and integrity.
CSA STAR	Yes	low	Securing creator profile updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//image

Endpoint Path	/api/brand/campaigns//image
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1159
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1158, 1158, 1159, 1159, 1165, 1165, 1166, 1166, 1170, 1170, 1158, 1158, 1165, 1165, 1166, 1166, 1170, 1170, 1158, 1158, 1165, 1165, 1166, 1166, 1170, 1170.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign images.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign images supports access control and integrity.
CSA STAR	Yes	low	Securing campaign image updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//budget

Endpoint Path	/api/brand/campaigns//budget
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1198
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1197, 1197, 1198, 1198, 1199, 1199, 1204, 1204, 1197, 1197, 1199, 1199, 1204, 1204, 1197, 1197, 1199, 1199, 1204, 1204.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign budgets.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign budgets supports access control and integrity.
CSA STAR	Yes	low	Securing campaign budget updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//requirements

Endpoint Path	/api/brand/campaigns//requirements
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1230
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1229, 1229, 1230, 1230, 1231, 1231, 1235, 1235, 1229, 1229, 1231, 1231, 1235, 1235, 1229, 1229, 1231, 1231, 1235, 1235.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign requirements.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign requirements supports access control and integrity.
CSA STAR	Yes	low	Securing campaign requirements updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//status

Endpoint Path	/api/brand/campaigns//status
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1259
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1258, 1258, 1259, 1259, 1260, 1260, 1264, 1264, 1258, 1258, 1260, 1260, 1264, 1264, 1258, 1258, 1260, 1260, 1264, 1264.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign status.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign status supports access control and integrity.
CSA STAR	Yes	low	Securing campaign status updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//view_threshold

Endpoint Path	/api/brand/campaigns//view_threshold
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1290
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1289, 1289, 1290, 1290, 1291, 1291, 1295, 1295, 1289, 1289, 1291, 1291, 1295, 1295, 1289, 1289, 1291, 1291, 1295, 1295.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign view thresholds.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign view thresholds supports access control and integrity.
CSA STAR	Yes	low	Securing campaign view threshold updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//deadline

Endpoint Path	/api/brand/campaigns//deadline
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1321
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1320, 1320, 1321, 1321, 1322, 1322, 1326, 1326, 1320, 1320, 1322, 1322, 1326, 1326, 1320, 1320, 1322, 1322, 1326, 1326.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures authorized users can update campaign deadlines.
ISO/IEC 27001	Yes	low	Token-based authentication for updating campaign deadlines supports access control and integrity.
CSA STAR	Yes	low	Securing campaign deadline updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/campaigns//pending-payouts

Endpoint Path	/api/brand/campaigns//pending-payouts
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1358
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1357, 1357, 1358, 1358, 1365, 1365, 1366, 1366, 1371, 1371, 1357, 1357, 1365, 1365, 1366, 1366, 1371, 1371, 1357, 1357, 1365, 1365, 1366, 1366, 1371, 1371.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures authorized users can view pending payouts for specific campaigns.
ISO/IEC 27001	Yes	low	Token-based authentication for viewing pending payouts supports access control and integrity.
CSA STAR	Yes	low	Securing access to pending payout information with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/profile

Endpoint Path	/api/brand/profile
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1455
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1454, 1454, 1455, 1455, 1456, 1456, 1460, 1460, 1454, 1454, 1456, 1456, 1460, 1460, 1454, 1454, 1456, 1456, 1460, 1460.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures brands can only view their own profile.
ISO/IEC 27001	Yes	low	Token-based authentication enforces access control for brand profile data.
CSA STAR	Yes	low	Securing brand profile data with tokens is a standard security measure.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/brand/profile

Endpoint Path	/api/brand/profile
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1482
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1481, 1481, 1482, 1482, 1483, 1483, 1487, 1487, 1481, 1481, 1483, 1483, 1487, 1487, 1481, 1481, 1483, 1483, 1487, 1487.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures brands can update only their own profile.
ISO/IEC 27001	Yes	low	Token-based authentication for brand profile updates supports access control and integrity.
CSA STAR	Yes	low	Securing brand profile updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/clip//view-count

Endpoint Path	/api/admin/clip//view-count
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1514
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1513, 1513, 1514, 1514, 1528, 1528, 1529, 1529, 1513, 1513, 1528, 1528, 1529, 1529, 1513, 1513, 1528, 1528, 1529, 1529.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures only administrators can update clip view counts.
ISO/IEC 27001	Yes	low	Token-based authentication for admin clip view count updates supports access control and integrity.
CSA STAR	Yes	low	Securing admin clip view count updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/campaign//update-views

Endpoint Path	/api/admin/campaign//update-views
HTTP Methods	PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1580
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1579, 1579, 1580, 1580, 1591, 1591, 1592, 1592, 1579, 1579, 1591, 1591, 1592, 1592, 1579, 1579, 1591, 1591, 1592, 1592.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated PUT operation ensures only administrators can update campaign view counts.
ISO/IEC 27001	Yes	low	Token-based authentication for admin campaign view count updates supports access control and integrity.
CSA STAR	Yes	low	Securing admin campaign view count updates with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/admin/analytics/campaign-performance/

Endpoint Path	/api/admin/analytics/campaign-performance/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1635
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1634, 1634, 1635, 1635, 1640, 1640, 1641, 1641, 1634, 1634, 1640, 1640, 1641, 1641, 1634, 1634, 1640, 1640, 1641, 1641.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures only administrators can view campaign performance analytics.
ISO/IEC 27001	Yes	low	Token-based authentication for accessing analytics data supports access control and integrity.
CSA STAR	Yes	low	Securing access to analytics data with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/auth/google-sync

Endpoint Path	/api/auth/google-sync
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1738
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1737, 1737, 1738, 1738, 1738, 1739, 1744, 1744, 1745, 1745, 1748, 1748, 1761, 1761, 1761, 1737, 1737, 1739, 1744, 1744, 1745, 1745, 1748, 1748, 1772, 1772, 1778, 1737, 1737, 1744, 1744, 1745, 1745, 1772, 1772, 1778, 1772, 1772, 1782.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures only authorized users can sync with Google.
ISO/IEC 27001	Yes	low	Token-based authentication for Google sync supports access control and integrity of linked accounts.
CSA STAR	Yes	low	Securing Google sync functionality with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /refresh

Endpoint Path	/refresh
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1816
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1815, 1815, 1816, 1816, 1821, 1821, 1822, 1822, 1815, 1815, 1821, 1821, 1822, 1822, 1815, 1815, 1821, 1821, 1822, 1822.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated refresh token endpoint is standard for maintaining session security.
ISO/IEC 27001	Yes	low	Secure handling of refresh tokens is part of session management, aligning with ISO 27001.
CSA STAR	Yes	low	Authenticated token refresh is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /logout

Endpoint Path	/logout
HTTP Methods	DELETE
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1832
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1831, 1831, 1832, 1832, 1836, 1836, 1831, 1831, 1836, 1836, 1831, 1831, 1836, 1836.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated logout ensures proper session termination.
ISO/IEC 27001	Yes	low	Secure logout mechanisms are essential for session management.
CSA STAR	Yes	low	Authenticated logout is a standard security practice for revoking access.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /api/health

Endpoint Path	/api/health
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\app.py : 1843
Authentication Required	No
Risk Severity	Low
CVSS Score	2.7

Authentication Analysis

This endpoint does not enforce authentication, which may expose it to unauthorized access depending on its functionality and the sensitivity of data processed.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

- **AUTH_MISSING** (low): Health check endpoint does not require authentication.

  Potential Attack Scenario:

  Generally acceptable for health checks, but if it reveals internal system status that could be leveraged by an attacker, it might pose a low risk.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Health check endpoints are typically public and do not require authentication.
ISO/IEC 27001	Yes	low	Publicly accessible health checks are standard and do not violate ISO 27001 principles.
CSA STAR	Yes	low	Publicly accessible health checks are common in cloud environments.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /create-deposit-order

Endpoint Path	/create-deposit-order
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 49
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 48, 48, 49, 49, 50, 50, 55, 55, 48, 48, 50, 50, 55, 55, 48, 48, 50, 50, 55, 55.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized deposit order creation.
ISO/IEC 27001	Yes	low	Token-based authentication for payment operations supports access control and integrity.
CSA STAR	Yes	low	Securing payment creation with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /verify-deposit

Endpoint Path	/verify-deposit
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 121
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 120, 120, 121, 121, 122, 122, 127, 127, 120, 120, 122, 122, 127, 127, 120, 120, 122, 122, 127, 127.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures only authorized users can verify deposits.
ISO/IEC 27001	Yes	low	Token-based authentication for deposit verification supports access control and integrity.
CSA STAR	Yes	low	Securing deposit verification with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /virtual-account

Endpoint Path	/virtual-account
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 192
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 191, 191, 192, 192, 193, 193, 198, 198, 191, 191, 193, 193, 198, 198, 191, 191, 193, 193, 198, 198.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures users can only retrieve their own virtual account information.
ISO/IEC 27001	Yes	low	Token-based authentication for virtual account details supports access control and integrity.
CSA STAR	Yes	low	Securing access to virtual account information with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /wallet-balance

Endpoint Path	/wallet-balance
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 243
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 242, 242, 243, 243, 244, 244, 247, 247, 242, 242, 244, 244, 247, 247, 242, 242, 244, 244, 247, 247.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures users can only view their own wallet balance.
ISO/IEC 27001	Yes	low	Token-based authentication for wallet balance retrieval supports access control and integrity.
CSA STAR	Yes	low	Securing access to wallet balance information with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /allocate-budget

Endpoint Path	/allocate-budget
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 269
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 268, 268, 269, 269, 270, 270, 275, 275, 268, 268, 270, 270, 275, 275, 268, 268, 270, 270, 275, 275.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized budget allocation.
ISO/IEC 27001	Yes	low	Token-based authentication for budget allocation supports access control and integrity.
CSA STAR	Yes	low	Securing budget allocation with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /reclaim-budget

Endpoint Path	/reclaim-budget
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 349
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 348, 348, 349, 349, 350, 350, 355, 355, 348, 348, 350, 350, 355, 355, 348, 348, 350, 350, 355, 355.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized budget reclamation.
ISO/IEC 27001	Yes	low	Token-based authentication for budget reclamation supports access control and integrity.
CSA STAR	Yes	low	Securing budget reclamation with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /distribute-to-creator

Endpoint Path	/distribute-to-creator
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 436
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 435, 435, 436, 436, 455, 455, 460, 460, 435, 435, 455, 455, 460, 460, 435, 435, 455, 455, 460, 460.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized distribution of funds to creators.
ISO/IEC 27001	Yes	low	Token-based authentication for creator payouts supports access control and financial integrity.
CSA STAR	Yes	low	Securing creator payouts with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator-withdraw

Endpoint Path	/creator-withdraw
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 560
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 559, 559, 560, 560, 578, 578, 583, 583, 559, 559, 578, 578, 583, 583, 559, 559, 578, 578, 583, 583.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can only withdraw from their own accounts.
ISO/IEC 27001	Yes	low	Token-based authentication for creator withdrawals supports access control and financial integrity.
CSA STAR	Yes	low	Securing creator withdrawals with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/payout-details

Endpoint Path	/creator/payout-details
HTTP Methods	POST, PUT
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 735
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 734, 734, 735, 735, 750, 750, 755, 755, 734, 734, 750, 750, 755, 755, 734, 734, 750, 750, 755, 755.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can save and update only their own payout details.
ISO/IEC 27001	Yes	low	Token-based authentication for saving payout details supports access control and data integrity.
CSA STAR	Yes	low	Securing payout details with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/payout-details

Endpoint Path	/creator/payout-details
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 823
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 822, 822, 823, 823, 828, 828, 833, 833, 822, 822, 828, 828, 833, 833, 822, 822, 828, 828, 833, 833.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can view only their own payout details.
ISO/IEC 27001	Yes	low	Token-based authentication for retrieving payout details supports access control and integrity.
CSA STAR	Yes	low	Securing access to payout details with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/verify-payout-details

Endpoint Path	/creator/verify-payout-details
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 873
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 872, 872, 873, 873, 878, 878, 883, 883, 872, 872, 878, 878, 883, 883, 872, 872, 878, 878, 883, 883.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can verify only their own payout details.
ISO/IEC 27001	Yes	low	Token-based authentication for verifying payout details supports access control and integrity.
CSA STAR	Yes	low	Securing payout details verification with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/withdrawals

Endpoint Path	/creator/withdrawals
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 933
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 932, 932, 933, 933, 940, 940, 945, 945, 932, 932, 940, 940, 945, 945, 932, 932, 940, 940, 945, 945.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can view only their own withdrawal history.
ISO/IEC 27001	Yes	low	Token-based authentication for withdrawal history supports access control and integrity.
CSA STAR	Yes	low	Securing access to withdrawal history with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/notifications/

Endpoint Path	/creator/notifications/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 995
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 994, 994, 995, 995, 999, 999, 1000, 1000, 1002, 1002, 994, 994, 999, 999, 1000, 1000, 1002, 1002, 994, 994, 999, 999, 1000, 1000, 1002, 1002, 1000, 1000, 1002, 1002, 1002, 1002, 1002, 1002.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can view only their own notifications.
ISO/IEC 27001	Yes	low	Token-based authentication for creator notifications supports access control and privacy.
CSA STAR	Yes	low	Securing access to creator notifications with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /transactions//

Endpoint Path	/transactions//
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1030
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1029, 1029, 1030, 1030, 1030, 1031, 1031, 1035, 1035, 1035, 1038, 1042, 1029, 1029, 1031, 1031, 1035, 1035, 1035, 1038, 1042, 1029, 1029, 1031, 1031, 1035, 1035, 1038, 1042, 1038, 1042, 1038, 1042.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures users can view only their own transaction history.
ISO/IEC 27001	Yes	low	Token-based authentication for transaction history supports access control and data privacy.
CSA STAR	Yes	low	Securing access to transaction history with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /refund-campaign

Endpoint Path	/refund-campaign
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1110
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1109, 1109, 1110, 1110, 1126, 1126, 1131, 1131, 1109, 1109, 1126, 1126, 1131, 1131, 1109, 1109, 1126, 1126, 1131, 1131.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized campaign refunds.
ISO/IEC 27001	Yes	low	Token-based authentication for campaign refunds supports access control and financial integrity.
CSA STAR	Yes	low	Securing campaign refunds with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /campaign-summary/

Endpoint Path	/campaign-summary/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1203
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1202, 1202, 1203, 1203, 1215, 1215, 1217, 1217, 1202, 1202, 1215, 1215, 1217, 1217, 1202, 1202, 1215, 1215, 1217, 1217.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures authorized users can view campaign summaries.
ISO/IEC 27001	Yes	low	Token-based authentication for campaign summaries supports access control and data integrity.
CSA STAR	Yes	low	Securing access to campaign summaries with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /calculate-earnings//

Endpoint Path	/calculate-earnings//
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1270
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1269, 1269, 1270, 1270, 1283, 1283, 1285, 1285, 1269, 1269, 1283, 1283, 1285, 1285, 1269, 1269, 1283, 1283, 1285, 1285.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures creators can calculate earnings only for their associated campaigns.
ISO/IEC 27001	Yes	low	Token-based authentication for earnings calculation supports access control and financial integrity.
CSA STAR	Yes	low	Securing earnings calculation with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /bulk-distribute

Endpoint Path	/bulk-distribute
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1364
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1363, 1363, 1364, 1364, 1384, 1384, 1389, 1389, 1363, 1363, 1384, 1384, 1389, 1389, 1363, 1363, 1384, 1384, 1389, 1389.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized bulk distribution of funds.
ISO/IEC 27001	Yes	low	Token-based authentication for bulk distribution supports access control and financial integrity.
CSA STAR	Yes	low	Securing bulk distribution with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /request-refund

Endpoint Path	/request-refund
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1535
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1534, 1534, 1535, 1535, 1548, 1548, 1553, 1553, 1534, 1534, 1548, 1548, 1553, 1553, 1534, 1534, 1548, 1548, 1553, 1553.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access prevents unauthorized refund requests.
ISO/IEC 27001	Yes	low	Token-based authentication for refund requests supports access control and integrity.
CSA STAR	Yes	low	Securing refund requests with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /refund-requests

Endpoint Path	/refund-requests
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1629
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1628, 1628, 1629, 1629, 1636, 1636, 1641, 1641, 1628, 1628, 1636, 1636, 1641, 1641, 1628, 1628, 1636, 1636, 1641, 1641.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures authorized users can view refund requests.
ISO/IEC 27001	Yes	low	Token-based authentication for refund requests supports access control and integrity.
CSA STAR	Yes	low	Securing access to refund requests with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /admin/approve-refund

Endpoint Path	/admin/approve-refund
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1696
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1695, 1695, 1696, 1696, 1709, 1709, 1714, 1714, 1695, 1695, 1709, 1709, 1714, 1714, 1695, 1695, 1709, 1709, 1714, 1714.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated POST operation ensures only administrators can approve refunds.
ISO/IEC 27001	Yes	low	Token-based authentication for approving refunds supports access control and financial integrity.
CSA STAR	Yes	low	Securing refund approval with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /admin/reject-refund

Endpoint Path	/admin/reject-refund
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1810
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1809, 1809, 1810, 1810, 1821, 1821, 1826, 1826, 1809, 1809, 1821, 1821, 1826, 1826, 1809, 1809, 1821, 1821, 1826, 1826.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated POST operation ensures only administrators can reject refunds.
ISO/IEC 27001	Yes	low	Token-based authentication for rejecting refunds supports access control and financial integrity.
CSA STAR	Yes	low	Securing refund rejection with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /refund-status/

Endpoint Path	/refund-status/
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1871
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1870, 1870, 1871, 1871, 1878, 1878, 1880, 1880, 1870, 1870, 1878, 1878, 1880, 1880, 1870, 1870, 1878, 1878, 1880, 1880.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures users can view status of their own refund requests.
ISO/IEC 27001	Yes	low	Token-based authentication for refund status supports access control and integrity.
CSA STAR	Yes	low	Securing access to refund status with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /admin/refund-audit-trail

Endpoint Path	/admin/refund-audit-trail
HTTP Methods	GET
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 1940
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 1939, 1939, 1940, 1940, 1947, 1947, 1939, 1939, 1947, 1947, 1939, 1939, 1947, 1947.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated access ensures only administrators can view the refund audit trail.
ISO/IEC 27001	Yes	low	Token-based authentication for audit trails supports access control and integrity.
CSA STAR	Yes	low	Securing access to audit trails with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

Endpoint Path /creator/revert-withdrawal

Endpoint Path	/creator/revert-withdrawal
HTTP Methods	POST
Source Location	..\D:\Downloads\Project\SecurityMongul\PulledCode_temp\f72882f4-802f-4518-855f-0513f4c0a125\Mipoe-Backend\routes\payments.py : 2019
Authentication Required	Yes (token)
Risk Severity	Low
CVSS Score	4.3

Authentication Analysis

This endpoint enforces authentication using a token-based mechanism. Authentication checks were detected at the following code locations: 2018, 2018, 2019, 2019, 2024, 2024, 2029, 2029, 2018, 2018, 2024, 2024, 2029, 2029, 2018, 2018, 2024, 2024, 2029, 2029.

Request Analysis

Content Type: unknown

No request fields were identified for this endpoint.

Response Analysis

Content Type: unknown

Status Codes:

Contains Sensitive Data: No

Identified Security Risks

No direct security risks were identified for this endpoint.

Compliance Impact

Regulation	Applicable	Risk Level	Reason
SOC 2	Yes	low	Authenticated POST operation ensures only authorized users can revert withdrawals.
ISO/IEC 27001	Yes	low	Token-based authentication for reverting withdrawals supports access control and financial integrity.
CSA STAR	Yes	low	Securing withdrawal reversion with tokens is a standard security practice.

Security Assessment Notes

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

References

METRICS SUMMARY

Total Findings: 6

Severity	Count
CRITICAL	**2**
HIGH	**0**
MEDIUM	**3**
LOW	**30**

DISCLAIMER

BUSINESS RISK ADVICE

Based on a comprehensive security analysis, here are the prioritized recommendations to enhance the security posture and mitigate identified risks.

Disable Flask Debug Mode in Production (Priority: Immediate)

Description: The Flask application is configured to run with debug mode enabled, exposing the Werkzeug debugger. This is a critical security vulnerability.

Why it Matters: Active debug mode allows remote code execution if the debug PIN is compromised, leading to full system compromise, data exfiltration, and denial of service.

Recommended Actions:

- Set `debug=False` in all production and staging environments.
- Utilize a production-ready WSGI server (e.g., Gunicorn, uWSGI) to serve the Flask application, preventing direct invocation of `app.run(debug=True)`.
- Implement dedicated, secure error logging and monitoring solutions instead of relying on the interactive debugger for operational issues.

Expected Outcome: Elimination of remote code execution risk, enhanced application stability, and prevention of sensitive information leakage through debug interfaces.

Remove Hardcoded JWT Token (Priority: Immediate)

Description: A full JSON Web Token (JWT) with sensitive details and an extended validity period is directly hardcoded within the `tasks.py` file.

Why it Matters: Direct exposure of a valid JWT token can lead to unauthorized access, impersonation of users, and sensitive data disclosure. Even if intended for testing, its presence in the codebase is a severe security flaw.

Recommended Actions:

- Immediately remove the hardcoded JWT token from `tasks.py` and any other locations in the codebase.
- Invalidate the exposed token and any associated user sessions in all environments where it may have been valid.
- Implement a secure secrets management solution (e.g., environment variables, dedicated secret stores, or cloud secrets managers) for all sensitive credentials and tokens, ensuring they are never hardcoded.
- Establish and enforce policies to exclude sensitive development/testing artifacts from version control and production builds.

Expected Outcome: Prevention of unauthorized access and data breaches stemming from compromised tokens, significantly reducing the attack surface related to sensitive credentials.

Enforce Authentication for Critical User Flow Endpoints (Priority: Immediate)

Description: The `/register`, `/login`, and `/request-password-reset` endpoints are missing proper authentication mechanisms.

Why it Matters: Lack of authentication on these critical endpoints allows unauthenticated users to create accounts, attempt arbitrary logins, and trigger numerous password reset requests. This enables brute-force attacks, credential stuffing, denial-of-service, and account takeover attempts, directly violating SOC 2, ISO/IEC 27001, and CSA STAR access control principles.

Recommended Actions:

- Implement robust authentication and authorization checks for `/register`, `/login`, and `/request-password-reset` endpoints.
- For `/register`, enforce mechanisms like CAPTCHA or multi-factor authentication for new account creation.
- For `/login`, implement strong password policies, rate limiting, account lockout mechanisms, and consider multi-factor authentication (MFA).
- For `/request-password-reset`, implement rate limiting and ensure tokens are single-use, time-limited, and sent via secure channels.
- Conduct a thorough review of access controls for all identity and access management related endpoints.

Expected Outcome: Secured user registration, login, and password reset processes, significantly reducing the risk of unauthorized access, account compromise, and denial-of-service attacks.

Enforce Request Timeouts for External API Calls (Priority: Short-Term)

Description: Multiple HTTP requests made to external services (e.g., Cashfree API) using the `requests` library do not specify a timeout parameter.

Why it Matters: Without defined timeouts, the application can hang indefinitely if an external service is slow or unresponsive, leading to resource exhaustion, application unresponsiveness, and potential denial-of-service (DoS) for legitimate users. This directly impacts the application's availability and resilience.

Recommended Actions:

- Add a `timeout` parameter to all `requests` calls to external services, specifying both a connect and read timeout (e.g., `timeout=(3, 30)` for 3 seconds connection and 30 seconds read).
- Conduct a comprehensive audit of all outbound HTTP requests to identify and remediate any missing timeouts.
- Implement retry mechanisms with exponential backoff for transient network errors to improve robustness without indefinite waiting.

Expected Outcome: Improved application resilience against slow or unresponsive third-party services, preventing application freezes and ensuring consistent availability of payment processing and other critical functionalities.

Standardize Error Handling and Logging (Priority: Short-Term)

Description: The application's API endpoints frequently return raw Python exception details (`str(e)`) in error responses, exposing internal system information to clients.

Why it Matters: Verbose error messages leak sensitive information such as file paths, database query specifics, internal library versions, and architectural insights, which attackers can leverage to map the attack surface and craft more targeted attacks. This violates the principle of least privilege in information exposure.

Recommended Actions:

- Modify all API endpoints to return generic, user-friendly error messages to the client for all unhandled exceptions (e.g., 'An unexpected error occurred. Please try again later.').
- Implement a centralized error handling mechanism or middleware to standardize error responses across the application.
- Ensure that detailed exception information, including stack traces, is logged internally on the server to a secure, aggregated logging system (e.g., ELK stack, Splunk, cloud logging services) for debugging and monitoring.
- Review existing logging configurations to ensure logs are secured and retained according to compliance requirements (SOC 2, ISO 27001).

Expected Outcome: Reduced attack surface by preventing internal details from reaching clients, enhanced security posture, and improved system debuggability through structured internal logging.

Conduct API Endpoint Exposure Audit (Priority: Mid-Term)

Description: Endpoints like `/api/campaigns` and `/api/campaigns/`, while identified as having low-risk missing authentication (potentially public), require explicit verification regarding the sensitivity of data they expose.

Why it Matters: Even if an endpoint is intended for public access, inadvertent exposure of sensitive or proprietary campaign details without authorization can lead to information leakage and competitive disadvantage, impacting confidentiality and privacy compliance (e.g., GDPR data minimization if PII is involved).

Recommended Actions:

- Perform a detailed data classification exercise for all data returned by publicly accessible endpoints.
- Explicitly confirm and document that all information exposed via unauthenticated endpoints is non-sensitive and intended for public consumption.
- Implement data masking or redaction for any potentially sensitive fields that might inadvertently be included in public responses.
- Regularly review API documentation and actual API responses to ensure that data exposure aligns with intended public access policies.

Expected Outcome: Guaranteed absence of sensitive data exposure on public endpoints, alignment with data privacy principles, and reduced risk of information leakage.

Enhance Secure Development Lifecycle (SDLC) Practices (Priority: Mid-Term)

Description: The presence of critical vulnerabilities like hardcoded secrets and debug mode in production indicates gaps in the current SDLC.

Why it Matters: Without integrating security early and consistently throughout the development process, vulnerabilities will continue to be introduced, leading to costly remediation cycles and persistent security risks that undermine compliance efforts.

Recommended Actions:

- Integrate Static Application Security Testing (SAST) tools into the CI/CD pipeline to automatically detect hardcoded secrets, misconfigurations, and other common vulnerabilities.
- Implement pre-commit hooks or peer code review processes specifically focused on identifying and preventing the introduction of sensitive data and insecure configurations into the codebase.
- Provide regular and mandatory application security training for all developers, emphasizing secure coding practices, OWASP Top 10, and secrets management.
- Establish clear security coding guidelines and standards that are regularly updated and enforced.

Expected Outcome: Proactive identification and prevention of security flaws, leading to a more secure codebase, reduced technical debt, and improved developer security awareness.

Implement Granular Authorization Checks (Priority: Long-Term)

Description: While authentication is present on most non-public endpoints, a comprehensive review of fine-grained authorization logic is essential.

Why it Matters: Even with authentication, users may be able to access or modify resources beyond their intended permissions (horizontal or vertical privilege escalation). Inadequate authorization leads to broken access control, a critical OWASP Top 10 vulnerability, directly impacting data integrity and confidentiality requirements for SOC 2 and ISO/IEC 27001.

Recommended Actions:

- Develop a detailed authorization matrix that maps user roles/types (e.g., Brand, Creator, Admin) to specific permissions for each API endpoint and resource.
- Implement a robust Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) system across the application.
- Ensure that authorization checks are performed at every access point to sensitive resources, not just at the controller/router level.
- Regularly audit and test authorization mechanisms (e.g., using dynamic application security testing - DAST) to confirm they function as intended.

Expected Outcome: Prevention of unauthorized data access and manipulation, ensuring users can only interact with resources they are explicitly permitted to, significantly strengthening the application's overall security posture.

Formalize Compliance Program and Documentation (Priority: Long-Term)

Description: The application's sector (Technology & SaaS) and anticipated compliances (SOC 2, ISO/IEC 27001, CSA STAR) require a formalized program for continuous adherence.

Why it Matters: Formal compliance is crucial for maintaining customer trust, enabling business operations, and avoiding legal and financial penalties. Ad-hoc security measures are insufficient to meet rigorous compliance standards for data protection, access control, and risk management.

Recommended Actions:

- Develop and implement formal security policies and procedures covering all aspects of information security, data handling, incident response, and risk management.
- Establish a regular internal audit schedule to assess compliance with defined policies and external regulations (SOC 2, ISO/IEC 27001, CSA STAR).
- Maintain comprehensive documentation of security controls, risk assessments, and compliance efforts.
- Designate clear roles and responsibilities for security governance and compliance within the organization.
- Work towards official certifications (e.g., ISO/IEC 27001) or attestations (e.g., SOC 2 Type 2) to demonstrate continuous compliance.

Expected Outcome: Achieved and maintained compliance with relevant industry and regulatory standards, enhanced organizational reputation, and reduced legal and financial risks associated with non-compliance.