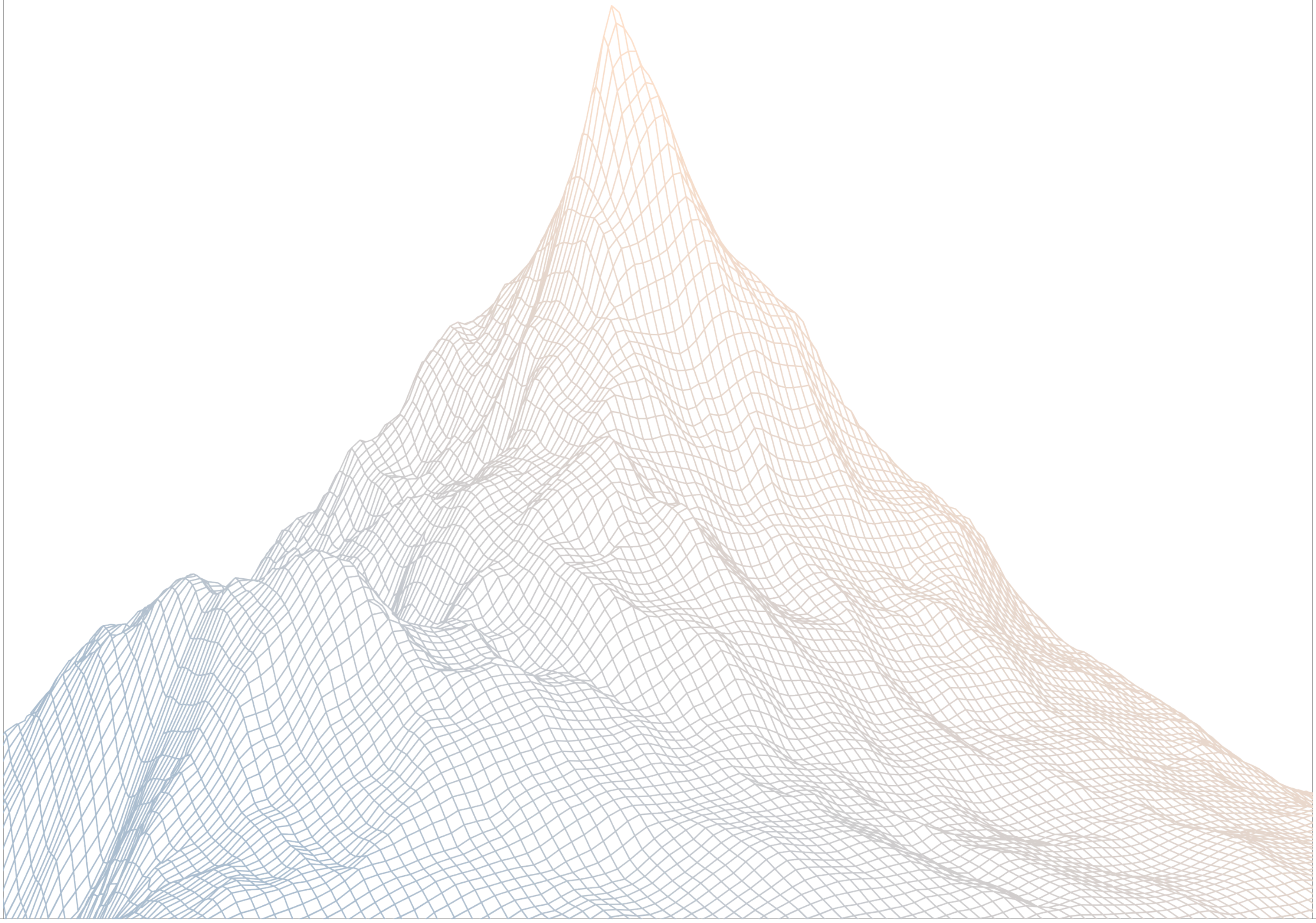
Zenith

Shogun

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: December 18th to January 13th, 2025

AUDITED BY: Jakub Heba
Victor Magnum
Spicymeatball

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Shogun

Shogun is the Integration Layer: a high-performance intent-centric platform that consolidates disparate blockchains into one unified application layer. With potentially hundreds or even thousands of chains being deployed, Shogun is the easiest way to get on-chain for both end-users and developers without worrying about the nuances of each chain.

2.2 Scope

The engagement involved a review of the following targets:

Target	shogun-bot-api
Repository	https://github.com/shogun-network/shogun-bot-api
Commit Hash	`43ab0aed0fbcd123bcac3d089e74898e25b86c0a`

Files	`API Security`
	`Logging and Monitoring`

Target	bot-login-backend
Repository	https://github.com/shogun-network/bot-login-backend
Commit Hash	`81e7a33ef6db2e5ff9cf3a36818e71d6649081fd`

Files	`Test for secure implementation of user login flows.`
	`Review backend access controls and ensure role-based access management.`
	`Verify secure storage of tokens in the backend (encryption, access control).`

Target	bot-oidc-server
Repository	https://github.com/shogun-network/bot-oidc-server
Commit Hash	`c92cd57885593b4ae51273cd7616af5fcbe25b91`

Files	`Validate identity verification processes within the OIDC flows.`
	`Check for secure token issuance, storage, and transmission.`
	`Assess the adherence to OIDC protocols`
	`Verify secure implementation of OIDC claims to avoid data leaks.`

Target	oidc-turnkey-issuer
Repository	https://github.com/shogun-network/oidc-turnkey-issuer
Commit Hash	`8f24b7211e29159a4006d216e05f1b8e54534cd4`

Files	`Identity and Access Management`
	`Protocol Implementation`

Target	bot-login-page
Repository	https://github.com/shogun-network/bot-login-page
Commit Hash	`737298c4ff9fa6e1910b4266ed052f750e2c6a74`

Files	`BOT API`
	`BOT Frontend`

Target	dextra-node
Repository	https://github.com/shogun-network/dextra-node
Commit Hash	`6028ed427cbfe6915402a72d8c293801d6ae58f4`

Files	`API Endpoint Security`
	`Data Transmission`
	`Dockerfile`
	`src/processors/tokenList.ts`
	`src/processors/routerApiKey.ts`

Target	MulticallRelayer-contract
Repository	https://github.com/shogun-network/MulticallRelayer-contract
Commit Hash	`00efc449bc0cabff6f2643b678433f979213f740`

Files	`TokenRecovery.sol`
	`ShogunMulticallV1.sol`
	`ShogunMulticallRelayerV1.sol`
	`ShogunLzComposerV1.sol`
	`ShogunExternalCallHandlerV1.sol`

2.3 Audit Timeline

December 18, 2025	Audit start
January 13, 2025	Audit end
January 15, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	4
Low Risk	9
Informational	1
Total Issues	**15**

3 Findings Summary

ID	Description	Status
H-1	Possible theft of OFT tokens from ShogunLzComposerV1	Resolved
M-1	Use of insecure cryptographic function (Math.random)	Resolved
M-2	Cryptographic salt misuse in Turnkey	Resolved
M-3	Unencrypted database communication	Resolved
M-4	Incorrect encryption of user identifiers	Resolved
L-1	Error message information disclosure	Acknowledged
L-2	Insufficient nonce validation in OIDC implementation	Acknowledged
L-3	Missing service dependency in OIDC server	Resolved
L-4	Missing error handling in routerApiKey.ts	Resolved
L-5	Unused shutdown timeout configuration	Acknowledged
L-6	Insecure database connection configuration defaults	Resolved
L-7	Returning HTTP 500 code should be avoided	Resolved
L-8	Potential Docker container escape in node	Resolved
L-9	Unused content type handler	Acknowledged
I-1	Misleading input validation message	Acknowledged

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Possible theft of OFT tokens from ShogunLzComposerV1

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- ShogunLzComposerV1

Description: The `ShogunLzComposerV1` contract facilitates processing LayerZero composable messages from the Stargate OFT:

```
    function receiveTokenTaxi(
        Origin calldata _origin,
        bytes32 _guid,
        address _receiver,
        uint64 _amountSD,
        bytes calldata _composeMsg
    ) external nonReentrantAndNotPaused onlyCaller(tokenMessaging) {
        --- SNIP ---

>>      bool success = _outflow(_receiver, amountLD);
        if (success) {
            _postOutflow(_amountSD);
            // send the composeMsg to the endpoint
            if (hasCompose) {
>>              endpoint.sendCompose(_receiver, _guid, 0, composeMsg);
            }
```

In this function, Stargate sends tokens to the receiver (in this case, `ShogunLzComposerV1`) via the `_outflow` function and then saves the composed payload in the `endpoint`. The `endpoint.lzCompose` call is expected to be performed in a subsequent transaction. However, this creates a potential time gap during which `ShogunLzComposerV1` holds unprocessed OFT tokens on its balance.

An attacker could exploit this vulnerability by executing their own Stargate composed message to initiate a `transfer` call on the OFT address, specifying it as the `callTarget`, effectively stealing tokens from the `ShogunLzComposerV1` contract:

```
    function executeCall_Internal(
        address callTarget,
        bytes calldata callData,
        uint256 amount
    ) public {
        if (msg.sender ≠ address(this)) revert InternalCallOnly();
        if (callTarget == address(this))
    revert ReentrancyGuardReentrantCall();

        uint256 msgValue = 0;
        if (bridgeToken ≠ address(0)) {
            IERC20(bridgeToken).safeTransfer(callTarget, amount);
        } else {
            msgValue = amount;
        }

>>      (bool success, bytes memory data) = callTarget.call{value:
    msgValue}(callData);
```

Recommendations

To prevent this type of attack, it is advised to explicitly prohibit the `bridgeToken` address from being used as a `callTarget`.

Shogun: Fixed with the following commit

Zenith: Verified.

4.2 Medium Risk

A total of 4 medium risk findings were identified.

[M-1] Use of insecure cryptographic function (Math.random)

`SEVERITY`: Medium	`IMPACT`: Medium
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- `src/processors/routerApiKey.ts:11`

Description

The use of `Math.random` for generating random numbers introduces multiple risks due to its lack of cryptographic security. The `Math.random` function is not designed to be secure for cryptographic purposes as it relies on a pseudo-random number generator (PRNG) with predictable outputs under certain conditions.

Attackers could potentially predict the sequence of random values generated, which poses significant risks in scenarios where these functions are used for generating passwords, session tokens, or other security-sensitive values.

Affected functions:

- `generateApiKey` - API key generation

References:

- https://www.blackduck.com/blog/pseudorandom-number-generation.html
- https://medium.com/@EsteveSegura/rolling-the-dice-on-security-the-pitfalls-of-using-math-random-in-javascript-3e891d8e4ef6

```
// Helper function to generate a unique API key
export function generateApiKey() {
  return Math.random().toString(36).substring(2, 15);
}
```

Recommendations

We recommend replacing Math.random with a cryptographically secure random number generator (CSPRNG), such as the Web Crypto API in JavaScript (crypto.getRandomValues). This ensures that the random values are unpredictable and suitable for security-sensitive use cases.

Shogun: Fixed with the following commit

Zenith: Verified

[M-2] Cryptographic salt misuse in Turnkey

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- `src/turnkey/encryptionHelpers.ts:64`

Description:

In `src/turnkey/encryptionHelpers.ts:64` the implementation uses the same value for both the password and salt in PBKDF2 key derivation. This significantly weakens the cryptographic security by making the salt more predictable and negating its purpose of preventing rainbow table attacks.

```
function aesDecrypt(encryptedData: string, salt: string): string {
  // Derive the key from the salt using PBKDF2
  const key = crypto.pbkdf2Sync(salt, salt, 100000, 32, 'sha256');

  let iv: Buffer;
  let encrypted: string;
```

Recommendations:

Implement proper salt generation using a cryptographically secure random number generator. Salt should be unique for each encryption operation and stored alongside the encrypted data.

Example of fix:

```
const salt = crypto.randomBytes(32);
const key = crypto.pbkdf2Sync(password, salt, 100000, 32, 'sha256');
```

Shogun:

- BOT API FIX: PR-623
- BOT BACKEND: PR-38

User don't have password and we are using DEFAULT_TURNKEY_ORGANIZATION_ID to encrypt new data

Zenith: Verified.

[M-3] Unencrypted database communication

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- `bot-login-backend-main/src/database/connect.ts`

Description: In `bot-login-backend-main/src/database/connect.ts`, the database connection is configured with SSL disabled (`ssl: false`), leaving database communication unencrypted. This exposes sensitive data to potential interception in man-in-the-middle attacks, particularly concerning given the nature of the data being stored and transmitted.

```
import { drizzle } from "drizzle-orm/postgres-js";
import postgres from "postgres";
import * as schema from "../drizzle/schema";
import { sql } from "drizzle-orm";

const connectionConfig = {
  host: process.env.PG_HOST || "localhost",
  port: parseInt(process.env.PG_PORT || "5432", 10),
  database: process.env.PG_DATABASE || "your_database_name",
  username: process.env.PG_USER || "your_username",
  password: process.env.PG_PASSWORD || "your_password",
  max: 10, // Set the maximum number of connections in the pool
  idle_timeout: 30, // Close idle connections after 30 seconds
  connect_timeout: 30, // Timeout after 30 seconds when connecting
  ssl: false,
};

const db = postgres(connectionConfig);

const client = drizzle(db, { schema });
const database_status = client
  .execute(sql`SELECT 1`)
  .then(() ⇒ "Online")
  .catch((error) ⇒ "Offline");
export { client as dbClient , database_status};
```

Recommendations

Enable SSL/TLS encryption for all database connections by setting appropriate SSL configuration. Implement proper certificate validation and consider using certificate pinning for additional hardening. Document the SSL configuration requirements in deployment documentation and implement connection testing to verify encrypted communication.

Shogun: Fixed with the following commit

Zenith: Verified.

[M-4] Incorrect encryption of user identifiers

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- `bot-login-backend-main/src/services/user.ts:37`

Description:

In `bot-login-backend-main/src/services/user.ts:37`, the user service contains a logical error where the same encrypted value is assigned to both `encrypted_sub_organization_id` and `encrypted_user_id` fields. While some of these values, as communicated with the Client, are not used currently, during future development it might lead to serious issues, if other than expected data will be placed in the above mentioned parameters.

```
public async user(req: Request, res: Response) {
    try {
      const result = userSchema.safeParse(req.body);
      if (!result.success) {
        return handleError(res, result.error);
      }

      const initData = this.getInitData(res);
      if (!initData?.user) {
        return handleError(res, new Error("Initial data not found."));
      }

      const userId = String(initData.user.id);
      const user = await this.db.query.users.findFirst({
        where: (users, { eq }) ⇒ eq(users.id, userId),
      });

      if (!user) {
        return handleError(res, new Error("User not found."));
      }

      if (!user.subUserId && !user.subOrganizationId) {
        const [encrypted_sub_organization_id, encrypted_user_id] =
          ([
            encryptData(result.data.subOrganizationId),
            encryptData(result.data.subOrganizationId),
          ]);

        await this.updateUser(userId, {
          subOrganizationId: encrypted_sub_organization_id,
          subUserId: encrypted_user_id,
          ethereumAddress: result.data.ethereumAddress,
          solanaAddress: result.data.solanaAddress,
        });

        try {
          const createWalletPromises = result.data.addresses.map((wallet) ⇒
            this.createWallet({
              id: wallet.address,
              userId: userId,
              network: wallet.type,
              privateKey: "",
              publicKey: wallet.address,
              address: wallet.address,
              label: "",
              updatedAt: new Date().toISOString(),
            })
          );

          await Promise.all([createWalletPromises]);
        } catch (error) {
          console.log(error);
        }

        return handleResponse(res, 200, { isNewUser: true });
      }

      return handleResponse(res, 200, { isNewUser: false });
    } catch (error) {
      return handleError(res, error);
    }
  }
```

Recommendationss

Correct the encryption implementation to properly encrypt both identifiers separately. Technical documentation should states directly which parameters are assigned to the encrypted versions.

Shogun: Fixed with the following commit

Zenith: Verified.

4.3 Low Risk

A total of 9 low risk findings were identified.

[L-1] Error message information disclosure

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- `src/turnkey/encryptionHelpers.ts:43`

Description:

In `src/turnkey/encryptionHelpers.ts:43` the application returns detailed error messages during decryption failures, potentially exposing sensitive information about the internal cryptographic implementation. This could aid attackers in understanding the system's internals and crafting more targeted attacks.

```
throw new Error(`Decryption failed:
    ${error instanceof Error ? error.message : 'UnknownError'}`);
```

Recommendations:

Implement generic error messages for user-facing responses while logging detailed errors securely for debugging purposes. Replace the current implementation with a standardized error response that doesn't leak implementation details.

Shogun: Acknowledged. We are using ERROR_MESSAGES and decodeErrorMessage function to use common errors (or default error message) in case of Turnkey/any other API errors

[L-2] Insufficient nonce validation in OIDC implementation

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- bot-login-backend-main/src/services/user.ts

Description:

The OIDC implementation accepts and processes nonces without proper validation or tracking of used values. This allows potential replay attacks as the same nonce can be reused multiple times. While the impact is mitigated by Telegram's user authentication, it still represents a deviation from OIDC security best practices.

```
export class UserService {
  constructor() {}


  public async signIn(req: Request, res: Response) {
    const { nonce } = req.body;

    const initData = this.getInitData(res);
    if (!initData?.user) {
      return handleError(res, new Error("Initial data not found."));
    }
    const userId = String(initData.user.id);
    const oidc_client = await OIDC_PROVIDER.Client.find(client.id);
    const claims = {
      sub: userId,
      nonce: bytesToHex(sha256(nonce)),
    };
    // Use the claims in the token generation
    const token : any = new OIDC_PROVIDER.IdToken(claims, { client:
    oidc_client });

    token.scope = "openid";
    // Issue the token with the claims
    const id_token = await token.issue({ use: "idtoken" });
    return handleResponse(res, 200, { id_token });
  }

  private getInitData(res: Response): InitDataParsed | undefined {
    return res.locals.initData;
  }
```

Recommendations:

Implement a nonce tracking mechanism that validates each nonce is used only once per user session. Store used nonces with their expiration timestamps and implement cleanup of expired entries.

Shogun: Acknowledged.

[L-3] Missing service dependency in OIDC server

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- src/controllers/index.ts

Description:

In the `src/controllers/index.ts` file, the `UserService` dependency is imported but the corresponding service file is missing from the repository. This is problematic as the application attempts to use a non-existent service. Such architectural inconsistency indicates incomplete code deployment or missing components, which could lead to service disruption and potential security implications if an incorrect or malicious service implementation is introduced.

```
import { Request, Response } from "express";
import { UserService } from "../services/user";

export class RouteController {
  private userService: UserService;
  constructor() {
    this.userService = new UserService();
  }

  signIn = async (req: Request, res: Response) => {
    return this.userService.signIn(req, res);
  };
}
```

Recommendations

Implement the missing `UserService` module in the services directory. The implementation should follow the authentication and authorization patterns expected by the OIDC server. If the functionality is not needed currently, the code should be refactored to not utilize the non-existing parts.

Shogun: Fixed with the following commit

Zenith: Verified.

[L-4] Missing error handling in `routerApiKey.ts`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target:

- `src/processors/routerApiKey.ts:65`

Description:

The GET `/keys` endpoint in the API router implementation currently lacks proper error handling mechanisms. The endpoint directly awaits the database operation and returns results without any error catching or handling logic. It might force the server to return HTTP 500 error in case of failure, returning potentially confidential configuration data, and not giving the users the proper and meaningful error message.

```
// Get all API keys
// curl http://localhost:3000/apiKey/keys
routerApiKey.get(`/keys`, roleCheckerUpsertListAll, async (req, res) => {
  const apiKeys = await db.listAllApiKey();
  res.json(apiKeys);
});
```

Recommendations

We recommend wrapping the database operation in a try-catch block, providing appropriate HTTP status codes and error messages to clients, implementing structured error logging for debugging purposes, and ensuring sensitive error details are never exposed in production environments.

The error handling should account for various failure scenarios including database connectivity issues, timeout errors, and permission problems. Additionally, the implementation should maintain consistency with error handling patterns used across other endpoints in the API.

Shogun: Fixed with the following commit

Zenith: Verified

[L-5] Unused shutdown timeout configuration

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- `src/config/index.ts`

Description:

The OIDC server in `src/config/index.ts` defines a `shutdownTimeout` configuration parameter but lacks the implementation to utilize it. Without proper shutdown handling, the application may terminate unexpectedly during deployment or maintenance, potentially leaving operations incomplete and data in an inconsistent state. This could affect both security and reliability of the service.

```
import 'dotenv/config';

export const config = {
  port: process.env.NEXT_PUBLIC_PORT || 3000,
  corsOptions: {
    origin: process.env.NEXT_PUBLIC_ALLOWED_ORIGINS?.split(',') || '*',
  },
  shutdownTimeout: 10000, // 10 seconds
};
```

Recommendations:

Implement proper shutdown handling using the configured timeout. Add a `SIGTERM` signal handler that initiates a graceful shutdown process, allowing ongoing operations to complete within the specified timeout period. This should include closing database connections, completing in-flight requests, and properly terminating all child processes.

Shogun: Acknowledged.

[L-6] Insecure database connection configuration defaults

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- bot-login-backend-main/src/database/connect.ts

Description:

In `bot-login-backend-main/src/database/connect.ts` the database connection configuration includes hardcoded default credentials when environment variables are not present. While these defaults appear to be placeholder values, their presence poses a risk of unsuccessful attempts to log in into the database and no proper error handling if environment variables are not available.

```
import { drizzle } from "drizzle-orm/postgres-js";
import postgres from "postgres";
import * as schema from "../drizzle/schema";
import { sql } from "drizzle-orm";

const connectionConfig = {
  host: process.env.PG_HOST || "localhost",
  port: parseInt(process.env.PG_PORT || "5432", 10),
  database: process.env.PG_DATABASE || "your_database_name",
  username: process.env.PG_USER || "your_username",
  password: process.env.PG_PASSWORD || "your_password",
  max: 10, // Set the maximum number of connections in the pool
  idle_timeout: 30, // Close idle connections after 30 seconds
  connect_timeout: 30, // Timeout after 30 seconds when connecting
  ssl: false,
};

const db = postgres(connectionConfig);

const client = drizzle(db, { schema });
const database_status = client
  .execute(sql`SELECT 1`)
  .then(() ⇒ "Online")
  .catch((error) ⇒ "Offline");
```

```
export { client as dbClient , database_status};
```

Recommendations

Remove all default credential values from the configuration. Implement strict environment variable validation at application startup. The application should fail to start if required database credentials are not properly configured, preventing any possibility of running with insecure default values.

Shogun: Fixed with the following commit

Zenith: Verified.

[L-7] Returning HTTP 500 code should be avoided

`SEVERITY:` Low	`IMPACT:` Low
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- `src/processors/tokenList.ts:18`

Description:

It has been found that in `src/processors/tokenList.ts` in the `tokenList` function, in case of an API error, code 500 is returned along with the error being passed to output.

This is problematic because such debugging responses may contain potentially sensitive information such as server configuration, its internal paths, or even cached API keys.

```
/**
 * Returns tokens with top volume during last 24 hours
 */
export const tokenList = async (req: Request, res: Response) ⇒ {
  try {
    const { chain } = req.query;
    if (!chain) {
      return res.status(400).send({ error: 'Chain parameter is required' });
    }

    const topTokensArray = await fetchTopTokens(chain as string);
    res.status(200).send(topTokensArray);
  } catch (error) {
    console.error("error", error);
    res.status(500).send({ error: (error as Error).message });
  }
};
```

Recommendations:

We suggest changing the logic of the `tokenList` function so that a generic error message is returned in case of an error.

Shogun: Fixed with the following commit

Zenith: Verified

[L-8] Potential Docker container escape in `node`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- `Dockerfile`

Description:

It was found that there is an absence of a non-root USER in the Dockerfile, which leaves the container susceptible to privilege escalation. If an attacker gains control over the host, he will have unrestricted privileges inside the container.

From container build to every future interaction, processes will run as root unless explicitly changed.

```
dockerfile.security.missing-user.missing-user
```

By not specifying a USER, a program in the container may run as 'root'. This is a security hazard. If an attacker can control a process running as root, they may have control over the container. Ensure that the last USER in a Dockerfile is a USER other than 'root'. Details: https://sg.run/Gbvn

| Autofix > USER non-root CMD ["node", "dist/index.js"] 28| CMD ["node", "dist/index.js"]

Recommendations:

It is recommended to modify Dockerfile to add a non-root user. By performing it, Docker will significantly reduces risk and restricts capabilities in case of a compromise. This adjustment will mitigate privilege risks, effectively reducing the attack surface.

Shogun: Fixed with the following commit

Zenith: Verified.

[L-9] Unused content type handler

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- bot-login-backend-main/src/middleware
- bot-oidc-server-main/src/middleware
- oidc-turnkey-issues-main/src/middleware

Description:

In multiple repositories, that is `bot-login-backend-main/src/middleware`, `bot-oidc-server-main/src/middleware` and `oidc-turnkey-issues-main/src/middleware`, in the `turnkey.ts` files, unused content type handler function exists in the codebase. While not directly causing security issues, dead code increases the attack surface and maintenance burden. It could also be accidentally enabled in future updates, potentially altering the application's content type handling behavior in unexpected ways.

```
export const handleContentType = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const contentType = req.get("Content-Type");

  if (contentType === "text/html" && req.method !== "GET") {
    req.headers["content-type"] = "application/json";
    console.warn(
      `Content-Type changed from ${contentType} to application/json`
    );
  }

  if (!contentType && req.method !== "GET") {
    req.headers["content-type"] = "application/json";
    console.warn("No Content-Type specified, defaulting to
    application/json");
  }
```

```
    next();
  };
```

Recommendations:

Remove the unused handler code. If content type manipulation is needed in the future, implement it as a new feature.

Shogun: Acknowledged.

4.4 Informational

A total of 1 informational findings were identified.

[I-1] Misleading input validation message

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Telegram bot

Description:

The bot's response during manual % of tokens choosing for transfer or swap indicates that percentage values from 0 to 100% are accepted for token transfers, while in reality, 0% transfers are not permitted. This inconsistency between the documented and implemented behavior could lead to user confusion.

Recommendations:

Update the input validation message to accurately reflect the accepted range of values. Ensure the information returned to the user clearly states that values must be between 1% and 100%.

Shogun: Acknowledged.