RD AUDITORS

GLITTER FINANCE BACKEND
AUDIT SUMMARY REPORT

Customer: Glitter Finance
Prepared on: 24th December 2022

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION.

THE REPORT CONTAINING CONFIDENTIAL INFORMATION CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

TABLE OF CONTENTS

Summary

A vulnerability assessment is a systematic review of security weaknesses in an information system. It evaluates if the system is susceptible to any known vulnerabilities, assigns severity levels to those vulnerabilities and recommends remediation or mitigation, if and whenever needed.

Source code analysis is basically automated code debugging. The aim is to find bugs and faults that may not be obvious to a programmer. It is meant to find faults like possible buffer overflows or untidy use of pointers and misuse of garbage collection functions, all of which may be exploitable by a hacker.

Code analyzers work using rules that tell them what to look for. With too little precision, an analyzer might spew out too many false positives and flood the user with useless warnings, while too much precision might take too long to finish; therefore has to be a balance.

There are two kinds of analyzers:

- Interprocedural - Detects patterns from one function to the next, and these patterns are correlated so that the analyzer can create a model and simulate execution paths.
- Intraprocedural - Focuses on pattern matching and depends on what kinds of patterns the user is looking for.

Interprocedural analyzers are more modern and more complex.

The assessment is performed from 12th to 24th December 2022.

Introduction

The penetration test is performed to identify loopholes in the application from a security perspective. The aim of this assessment is to discover the vulnerabilities present in the user-facing platform, which can pose an information security risk.

Overall, we were able to achieve the goals of the assessment and identify vulnerabilities in the target environment within the time window. There are several findings during the assessment for which the details will be provided in the findings section.

Source code analysis is the automated testing of source code for the purpose of debugging a computer program or application before it is distributed or sold. Source code consists of statements created with a text editor or visual programming tool and then saved in a file. The source code is the most permanent form of a program, even though the program may later be modified, improved, or upgraded.

A major advantage of this method is the fact that it does not require developers to make educated guesses at situations likely to produce errors. Other advantages include eliminating unnecessary program components and ensuring that the program under test is compatible with other programs likely to be run concurrently.

Vulnerability Assessment

Executive Summary

The purpose of this vulnerability scan is to gather the information about the network and data. We found no vulnerabilities during this assessment.

High	Medium	Low	Info
0	0	0	0

Code Coverage Testing

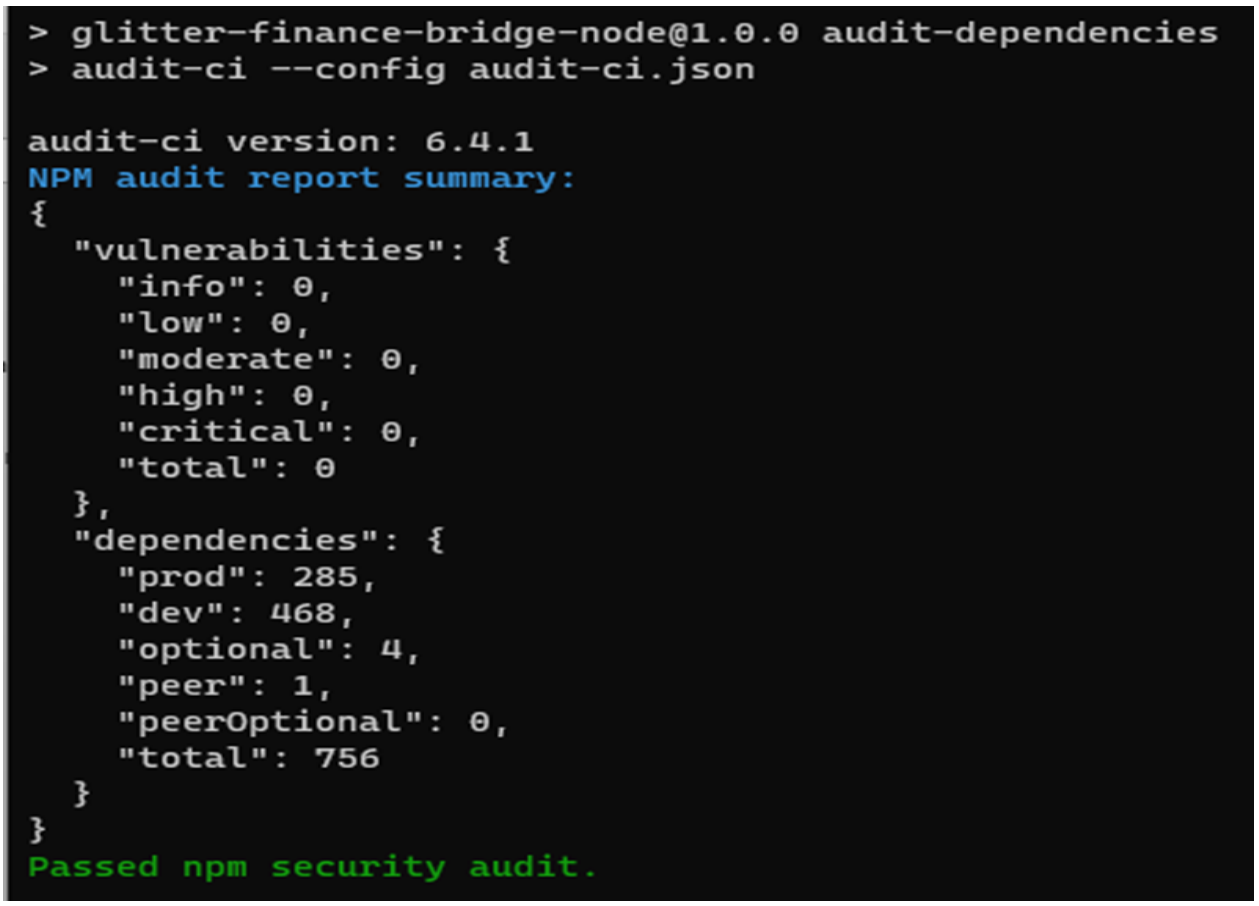
Code coverage testing is a software testing technique that measures the extent to which a given set of tests exercises the code of a program. It is used to determine how much of the code is executed when the tests are run and can help identify areas of the code that are not being adequately tested.

```
----------|---------|----------|---------|---------|-------------------
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------|---------|----------|---------|---------|-------------------
All files |       0 |        0 |       0 |       0 |
----------|---------|----------|---------|---------|-------------------
```

```
23 files checked.
testMatch:  - 0 matches
testPathIgnorePatterns: \\node_modules\\ - 23 matches
testRegex: (\\__tests__\\.*|\.(test|spec))\.(ts|tsx)$ - 0 matches
```

Dependency Analysis

A security audit is an assessment of dependencies for security vulnerabilities. Security audits help you protect your users by finding and fixing known vulnerabilities in dependencies that could cause data loss, service outages, unauthorized access to sensitive information, amongst other issues.



Code Review

1. Empty block statement

File name - services/backend/src/admin_connection.ts:-

```
78              //On Error
79              webSocket.onerror = (error) => {
80                  if (this._isConnected) {
81                      logger.logLine();
82                      logger.error('Encountered an error with Admin server: ' + error)
83                      logger.logLine();
84                  } else {
85
86                  }
87              }
```

Issue type - The else part is empty here. Most of the time a block of code is empty when a piece of code is really missing. So such an empty block must be either filled or removed.

2. Remove these unused imports

File name - services/backend/src/circle.ts

```
1  import { getNetwork } from "./config";
2  import { Chain, Circle, CircleEnvironments } from "@circle-fin/circle-sdk";
3  import { NETWORK } from "./type";
4
```

File name - services/backend/src/common.ts

```
1  import { v4 as uuidv4 } from 'uuid';
2  import { BusinessRecipientAddressObject, Chain, Circle, Transfer, TransferDestinationWalletLocation, Tr
3  import { fromTransactionExists, saveFromTransaction, saveToTransaction } from "@services/database/src/h
4  import { Network } from "@services/database/src/handle/type";
5  import { Routing, Sleep } from "glitter-bridge-common-dev";
6  import { bridgeFee, circleConfig } from "./config";
7  import { sleepPerAPI } from "./sync";
8  import { DepositNote } from "./type";
9  import { Logger } from 'glitter-bridge-common-dev/dist/src/utils/logger';
10 import { GlitterBridgeSDK } from 'glitter-bridge-sdk-dev';
11 import * as util from 'util';
12
```

File name - services/backend/src/config/index.ts

```
import { config } from "dotenv";
import { NETWORK } from "../type";
import { Chain, Circle, CircleEnvironments } from "@circle-fin/circle-sdk";
import fs from 'fs';
import algosdk from "algosdk";
import { clusterApiUrl, Connection, Keypair } from "@solana/web3.js";
import {SolanaUtils} from "glitter-bridge-solana-dev/dist/src/utils"
```

File name - services/backend/src/solana.ts

```
import bs58 from "bs58";
import { BusinessRecipientAddressObject, Circle, Transfer, TransferDetailedTransfer } from "@circle
import {
    saveToTransaction,
} from "@services/database/src/handle";
import * as solana from "@solana/web3.js";
import { Network } from "@services/database/src/handle/type";
import { getJsonRPC } from "./utils";
import { circleConfig, getNetwork, solanaConfig } from './config';
import { ReleaseFunctionParams, USDCBridgeCommon } from "./common";
import { Logger, Routing, RoutingString } from "glitter-bridge-common-dev";
import { DepositNote } from "./type";
import { GlitterBridgeSDK } from "glitter-bridge-sdk-dev";
import { sendAndConfirmTransaction, Transaction, TransactionInstruction } from '@solana/web3.js';
import { PublicKey } from '@solana/web3.js';
import * as splToken from "@solana/spl-token";
import { getMint, getOrCreateAssociatedTokenAccount, TOKEN_PROGRAM_ID } from '@solana/spl-token';
import { SolanaAccounts } from "glitter-bridge-solana-dev";

```

File name - services/backend/src/utils.ts

```
import { Chain } from '@circle-fin/circle-sdk';
import { Network } from '@services/database/src/handle/type';
import fs from 'fs';
import { getNetwork } from "./config";
import { NETWORK } from './type';

```

Issue type - There's no reason to import modules you don't use; and every reason not to do so needlessly increases the load.

3. Refactor this function to reduce its Cognitive Complexity from 20 to the 15 allowed.

File name - services/backend/src/sync.ts

Issue type - Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain.

4. Refactor this function to reduce its Cognitive Complexity from 33 to the 15 allowed.

File name - services/backend/src/common.ts

Issue type - Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain.

5. 'getTransaction' is deprecated. Instead, call `getTransaction` using a `GetVersionedTransactionConfig` by setting the `maxSupportedTransactionVersion` property.

File name - services/backend/src/solana.ts

```
39        //Get constants
40        const client = glitter_bridge.solana?.client;//    new solana.Connection
41        if (!client) {
42            logger.error("solana client not initialized");
43            process.exit();
44            return;
45        }
46        const transactionHash = transfer.transactionHash!;
47        const transaction = await client.getTransaction(transactionHash!);
48
```

Issue type - Once deprecated, classes, interfaces, and their members should be avoided, rather than used, inherited, or extended. Deprecation is a warning that the class or interface has been superseded, and will eventually be removed. The depreciation period allows you to make a smooth transition away from the aging, soon-to-be-retired technology.

6. This assertion is unnecessary since it does not change the type of the expression.

File name - services/backend/src/solana.ts

```
58          //Get deposit note
59          let depositNote
60          for (let i = 0; i < transaction.transaction.message.instructions.length; i++) {
61              const data_bytes = (bs58.decode(transaction.transaction.message.instructions[i].data) || "{
62              try {
63                  const object = JSON.parse(Buffer.from(data_bytes).toString('utf8'))
64                  if (object.system && object.date) {
65                      depositNote = object;
66                  }
67              } catch { }
68          }
```

Issue type - The TypeScript compiler automatically casts a variable to the relevant type inside conditionals where it is possible to infer the type (because the type of, the instance of, etc was used). This compiler feature makes casts and not-null assertions unnecessary.

7. 'transaction' is already declared in the upper scope.

File name - services/database/src/handle/index.ts

```
29  export const saveFromTransaction = async (fromTransaction: FromTransaction, toNetwork: Network, toA
30    try {
31      const dataSource = await getConnectedDataSource();
32      const transactionRepostiroy: Repository<Transaction> =
33        dataSource.getRepository(Transaction);
34      try {
35        const transaction = await transactionRepostiroy.findOne({
```

Issue type - Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another.

8. 'connect' is deprecated. use. initialize method instead.

File name - services/database/src/orm.config.ts

```
41    let dataSource: DataSource;
42    export const getConnectedDataSource = async () => {
43      if (dataSource) {
44        return dataSource;
45      }
46
47      dataSource = getConnectionSource();
48      await dataSource.connect();
49      return dataSource;
50    };
51
```

Issue type - Once deprecated, classes, interfaces, and their members should be avoided, rather than used, inherited, or extended. Deprecation is a warning that

the class or interface has been superseded, and will eventually be removed. The depreciation period allows you to make a smooth transition away from the aging, soon-to-be-retired technology.

9. Rename class "migration1667920844815" to match the regular expression ^[A-Z][a-zA-Z0-9]*$.

File name - services/database/src/migrations/1667920844815-migration.ts

```
1   import { MigrationInterface, QueryRunner } from "typeorm";
2
3   export class migration1667920844815 implements MigrationInterface {
4       name = 'migration1667920844815'
5
6       public async up(queryRunner: QueryRunner): Promise<void> {
7           await queryRunner.query(`CREATE TABLE "transaction" ("id" uuid NOT NULL DEFAULT uuid_genera
8       }
9
10      public async down(queryRunner: QueryRunner): Promise<void> {
11          await queryRunner.query(`DROP TABLE "transaction"`);
12      }
13
14  }
15
```

Issue type - Shared coding conventions allow teams to collaborate effectively This rule allows checking that all class names (and interfaces for TypeScript) match a provided regular expression.

10. Unsafe IP used.

File name - services/backend/src/admin_connection.ts

```
48      //Websocket
49      public static async createWebSocket(glitter_bridge:GlitterBridgeSDK,logger: Logger) {
50
51          const host = "10.114.0.3"
52          const port = 3040;
53          const endpoint = "admin";
54          const subscriberUrl = `ws://${host}:${port}/${endpoint}`;
55
```

Issue type - The disclosed IP address is sensitive. Can give information to an attacker about the network topology.

Code Analysis

1. Code Analysis

prettier - the Prettier tool itself;

eslint-config-prettier - rules for the Prettier code formatter;

eslint-plugin-prettier - turns off ESLint rules which might conflict with Prettier;

prettier-eslint - allows ESLint to auto-fix formatting issues in your code;

Steps:

Used bellow dependencies for Core TypeScript ESLint support

- npm install --save-dev @typescript-eslint/parser
- npm install --save-dev @typescript-eslint/eslint-plugin

Used bellow dependencies for extra code quality rules

- npm install --save-dev eslint-plugin-import
- npm install --save-dev eslint-plugin-sonarjs

Used bellow dependencies for prettier and eslint integration

- npm install --save-dev prettier
- npm install --save-dev eslint-config-prettier
- npm install --save-dev eslint-plugin-prettier
- npm install --save-dev prettier-eslint

Created configuration file '.eslintrc.js '. Added below script in package json file.

- "lint": "eslint './src/**/*.{tsx,ts}'",
- "lint-fix": "eslint './src/**/*.{tsx,ts}' --fix"

Used bellow dependencies for code coverage report generate

- npm install --save-dev jest @types/jest
- npm install --save-dev ts-jest jest-sonar-reporter

Created jest configuration file ‘jest.config.js’. Added below script in package json file to generate a report in the coverage folder.

- "jestSonar": {
    - "reportPath": "coverage",
    - "reportFile": "test-reporter.xml",
    - "indent": 4 }

Added below script in package json file for run test.

- "test": "jest --forceExit --detectOpenHandles --coverage"

Run the test using the ‘npm test’ command.

2. Dependency Audit / Supply Chain Analysis

Used Audit-ci for analysis dependencies.

- npm install --save-dev audit-ci

Created configuration file ‘audit-ci.json’. Added below script in package json file.

- "audit-dependencies": "audit-ci --config audit-ci.json"

Start audit using ‘npm run audit-dependencies’ command.

3. Used sonarqube for vulnerability assessment, code quality.

Created ‘docker-compose.sonar.yml’ file run application test. Added below code in yml file.

```
version: '3'

services:

  sonarqube:

      container_name: sonarqube

      image: sonarqube:latest
```

```
ports:
- '9000:9000'
- '9092:9092'
```

Created config file for sonarqube application configuration. Added below code.

```
sonar.projectKey=secure-typescript-boilerplate
sonar.projectName=secure-typescript-boilerplate
sonar.projectVersion=1.0
sonar.language=ts
sonar.sources=src
sonar.sourceEncoding=UTF-8
sonar.exclusions=src/**/*.test.ts
sonar.test.inclusions=src/**/*.test.ts
sonar.coverage.exclusions=src/**/*.test.ts,src/**/*.mock.ts,node_modules/*,coverage/lcov-report/*
sonar.typescript.lcov.reportPaths=coverage/lcov.info
sonar.testExecutionReportPaths=coverage/test-reporter.xml
```

Edited the code as per the project folder.

Used below dependency for run sonarqube test.

- npm install --save-dev sonarqube-scanner

Created config file ‘sonar.project.js’.

```
const sonarqubeScanner = require('sonarqube-scanner');
sonarqubeScanner(
  { serverUrl: process.env.SONAR_SERVER || 'http://localhost:9000',
    token: process.env.SONAR_TOKEN || '',
    options: {}},
```

```
() => {
        console.log('>> Sonar analysis is done!');});
```

Used below dependency for uploading the test data in local sonarqube server without failing the test.

- npm install --save-dev sonarqube-verify

Added script in package json file.

- "sonar": "sonarqube-verify"

Active docker using below command.

- docker-compose -f docker-compose.sonar.yml up -d

Run analysis using the ‘npm run sonar’ command.

Tested the application several times using this process with a different scenario.

Disclaimers

The associated code/URLs given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues regarding the penetration test, details of which are disclosed in this report.

Because the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the test.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of websites.

Email: info@rdauditors.com

Website: www.rdauditors.com