GLITTER FINANCE

BRIDGE NODE SECURITY

AUDIT REPORT

Customer: Glitter Finance
Prepared on: 11th July 2023

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND THE 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

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.

Database vulnerability testing, also known as database penetration testing or database security assessment, is the process of evaluating the security of a database system to identify potential vulnerabilities and weaknesses. It involves simulating real-world attack scenarios and assessing the effectiveness of security measures implemented in the database.

The goal of database vulnerability testing is to uncover vulnerabilities that could be exploited by attackers to gain unauthorised access, steal sensitive data, or disrupt the normal functioning of the database. It helps organisations identify and mitigate security risks by identifying weaknesses in the database infrastructure, configuration, access controls, and data handling processes.

During the testing process, various techniques and tools are used to probe the database for vulnerabilities, such as SQL injection, insecure configurations, weak authentication mechanisms, privilege escalation, and insecure data storage practices. The testing may involve both automated scanning tools and manual techniques to thoroughly assess the security posture of the database.

The results of a database vulnerability test typically include a comprehensive report that highlights identified vulnerabilities, their potential impact, and recommendations for remediation. This information can then be used by organisations to prioritise and address the identified security weaknesses, thereby improving the overall security of their database infrastructure.

The assessment is performed from 28th June to 11th July 2023.

Introduction

Penetration testing checks your organisation's web-facing assets for security vulnerabilities.

A successful pen test not only identifies the vulnerabilities but also finds different ways to exploit them and anticipates the impact on the system.

A database vulnerability refers to a weakness or flaw in a database system that can be exploited by attackers to gain unauthorised access, manipulate data, or disrupt the normal functioning of the database. These vulnerabilities can occur due to various reasons, such as software bugs, configuration errors, inadequate security measures, or lack of regular updates.

Exploiting a database vulnerability can have severe consequences, including unauthorised access to sensitive information, data breaches, data loss or corruption, and even unauthorised control over the entire database system. Attackers may use various techniques, such as SQL injection, cross-site scripting (XSS), buffer overflows, or brute-force attacks to exploit these vulnerabilities.

To mitigate database vulnerabilities, it is essential to follow security best practices, such as implementing strong access controls, regularly patching and updating the database software, employing secure coding practices, conducting regular security audits, and monitoring the system for any suspicious activities. Additionally, using encryption for sensitive data, implementing strong password policies, and performing regular backups can also help protect against potential vulnerabilities and their potential impacts.

Penetration testing (pen test) is a simulated cyberattack that helps organisations identify and fix security vulnerabilities in their web-facing assets. A successful pen test not only identifies vulnerabilities but also finds different ways to exploit them and anticipates the impact on the system.

Vulnerability Assessment

Executive Summary

The purpose of this vulnerability scan is to gather the information about the network and data. We found 4 low vulnerabilities during this assessment.

High	Medium	Low	Info
0	0	4	0

Low Vulnerabilities

1. [CVE-2022-25883] CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')

Risk Description: Versions of the package semver before 7.5.2 are vulnerable to Regular Expression Denial of Service (ReDoS) via the function new Range when untrusted user data is provided as a range.

2. [CVE-2023-34459] CWE-354: Improper Validation of Integrity Check Value

Risk Description: OpenZeppelin Contracts is a library for smart contract development. Starting in version 4.7.0 and prior to version 4.9.2, when the `verifyMultiProof`,`verifyMultiProofCalldata`,`procesprocessMultiProof`,or`processMultiProofCalldat` functions are in use, it is possible to construct Merkle trees that allow forging a valid multi proof for an arbitrary set of leaves.

3. [CVE-2022-25883] CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')

Risk Description: Versions of the package semver before 7.5.2 are vulnerable to Regular Expression Denial of Service (ReDoS) via the function new Range when untrusted user data is provided as a range.

4. [CVE-2016-20018] CWE-89: Improper Neutralisation of Special Elements used in an SQL Command ('SQL Injection')

Risk Description: The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralise or incorrectly neutralise special elements that could modify the intended SQL command when it is sent to a downstream component.

Code Coverage Testing

Executive Summary

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.

Grade	Info	Lines of Code	Analysed Files
Good	**1**	**15,466**	**139**

1. **File Path:** src/features/logger/infra/pino-logger.spec.ts
   Type: UNUSED_IMPORT
   Details: Imported binding '**Readable**' is not used.

Dependency Analysis

Executive Summary

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, unauthorised access to sensitive information, amongst other issues.

2 Moderate vulnerabilities found.

1. @openzeppelin/contractsImproper Input Validation

- Introduced through: glitter-bridge-token-node@0.1.0 › @glitter-finance/sdk-core@1.2.0-52 › glitter-evm-contracts@1.0.33 › @openzeppelin/contracts@4.8.3
- Introduced through: glitter-bridge-token-node@0.1.0 › @glitter-finance/sdk-server@1.0.1-34 › @glitter-finance/sdk-core@1.2.0-52 › glitter-evm-contracts@1.0.33 › @openzeppelin/contracts@4.8.3

@openzeppelin/contracts is a library for contract development. Affected versions of this package are vulnerable to Improper Input Validation when the verifyMultiProof, verifyMultiProofCalldata, procesprocessMultiProof, or processMultiProofCalldat functions are in use, it is possible to construct merkle trees that allow forging a valid multiproof for an arbitrary set of leaves. A contract may be vulnerable if it uses multiproofs for verification and the merkle tree that is processed includes a node with value 0 at depth 1 (just under the root). This could happen inadvertently for balanced trees with 3 leaves or less, if the leaves are not

hashed. This could happen deliberately if a malicious tree builder includes such a node in the tree. A contract is not vulnerable if it uses single-leaf proving (verify, verifyCalldata, processProof, or processProofCalldata), or if it uses multiproofs with a known tree that has hashed leaves. Standard merkle trees produced or validated with the @openzeppelin/merkle-tree library are safe.

2. semverRegular Expression Denial of Service (ReDoS)

- Introduced through: glitter-bridge-token-node@0.1.0 › @safe-global/safe-core-sdk@3.3.2 › semver@7.3.8
- Introduced through: glitter-bridge-token-node@0.1.0 › @glitter-finance/sdk-core@1.2.0-52 › tronweb@5.1.0 › semver@5.7.1
- Introduced through: glitter-bridge-token-node@0.1.0 › @safe-global/safe-core-sdk@3.3.2 › @safe-global/safe-core-sdk-utils@1.7.2 › semver@7.3.8

semver is a semantic version parser used by npm.
Affected versions of this package are vulnerable to Regular Expression Denial of Service (ReDoS) via the function new Range, when untrusted user data is provided as a range.

Code Review

1. **Use of Hardcoded Credentials**

Do not hardcode passwords in code. Found hardcoded password used.

```
23    process.env.POSTGRES_DB = "glitter_docker"
24    process.env.POSTGRES_OPTIONS = "sslmode=verify-full"
25    const mockCredentials = {
26      username: "glitter_docker",
27      password: "glitter_docker",
```

File name: in src/database/client.spec.ts file.

Code Analysis Process

1. **Source 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 followed:

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

Configuration file '.eslintrc.js '. Add below script in package json.

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

Dependencies for generating code coverage report.

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

Jest configuration file 'jest.config.js'. Add 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.

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_module
s/*,c overage/lcov-report/*
sonar.typescript.lcov.reportPaths=coverage/lcov.info
sonar.testExecutionReportPaths=coverage/test-reporter.xml
```

Updated the code as per the project folder.

Added below dependency for run sonarqube test.

- npm install --save-dev sonarqube-scanner

Added script in package json file.

- "sonar": "sonarqube-verify"

Active docker using below command.

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

Start analysis using the ‘npm run sonar’ command.
Test the application multiple times using this process with multiple scenarios.

Client Response

Vulnerability Assessment

1. Not applicable. The backend validators are isolated systems that do not process user input.
2. Not applicable. The backend validators do not make use of Merkle trees.
3. Not applicable. The backend validators are isolated systems that do not process user input.
4. Not applicable. The backend validators do not process user input, they are not susceptible to SQL injection.

Dependency Analysis

1. Not applicable. The backend validators do not make use of Merkle trees.
2. Not applicable. The backend validators are isolated systems that do not process user input.

Code Review

1. As the suffix implies (spec.ts) this file contains a collection of unit tests. In order to run unit tests, a local image of PostgreSQL is spun up by Docker, with a set of mock credentials. Mock credentials for local development do not pose a security risk. This code never runs in production. Mock credentials are never used on the production database either.

Disclaimers

RD Auditors Disclaimer

The associated code/URLs given for audit have been analysed 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 as 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