DECENTRALIZED APPLICATION CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Allbridge

Date: February 9th, 2023

ver. 3

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Decentralized Application Code Review and Security Analysis Report for Allbridge.
Approved By	Andrew Matiukhin | CTO at Hacken OÜ
Type	Server, Bridge Oracle
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	allbridge.io
Timeline	16 JANUARY 2023 - 09 FEBRUARY 2023
Changelog	27 JANUARY 2023 - Review[1] 02 FEBRUARY 2023 - Remediation[2] 09 FEBRUARY 2023 - Remediation[3]

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Allbridge (Customer) to conduct a Decentralized Application Code Review and Security Analysis. This report presents the security assessment findings of the Customer's applications.

Scope

The scope of the project is review and security analysis of applications in the following repositories:

1. https://github.com/allbridge-io/bridge-stellar-server
   - **Platform:** TypeScript (Node.js)
   - **Commit:** e113f0908441b24b77c64e4d85e7659154565d1a
2. https://github.com/allbridge-io/stellar-bridge-validator
   - **Platform:** TypeScript (Node.js)
   - **Commit:** c46a43aab88e99f41b780feafdcafb82a61189e0

These applications were scanned for commonly known and more specific issues. Considered items include but are not limited to:

- Overconfidence in a node (or node provider)
- Failure to account for blockchain branching out
- Incorrect validation of ENS records
- Weak authentication via message signing
- Unsafe private key storage
- XSS/SQL injections from the blockchain data
  Misuse of checksum addresses
- Blockchain data inconsistency
- Incorrect integration with a smart contract and/or blockchain platform
- Usage of wrong data types
- Application architecture
- Repository consistency
- Code style consistency
- Deprecated, vulnerable, or outdated Web3 libraries

System Overview

We have identified and reviewed the following interactions between the applications and blockchain platforms.

stellar-bridge-validator

This is a bridge oracle. It verifies the transfers made to the bridge account on the Stellar blockchain and provides the signatures that prove that.

It has a single public endpoint - GET */sign/$TRANSACTION_ID*. It searches for a transaction with the given identifier, collects all the relevant information, checks whether the transaction was already claimed, and provides a signature that proves this transaction was made on the Stellar blockchain.

bridge-stellar-server

This is a bridge oracle. It signs and broadcasts transactions to the Stellar and NEAR blockchains. Its API has public and private endpoints protected by a secret key.

Public endpoints are:

- POST */unlock*
  It validates the signatures and sends a transaction creating a transfer on the NEAR blockchain.
- GET */unlock/$SOURCE/$LOCK_ID*
  It checks if the transfer was already claimed.
- GET */token/$TOKEN_ADDRESS/$SYMBOL*
  It provides information about the given token.
- GET */balance/$ADDRESS*
  It provides the XLM balance for the given address.
- GET */fee/$TOKEN_ADDRESS/$SYMBOL*
  It provides fee-related information about the given token.

- POST */wallet/transaction*
- POST */wallet/create-line*
  These endpoints build a Stellar transaction with the given arguments and return it XDR-encoded.

Private endpoints are:

- POST */sign*
  It builds a payment transaction and provides the signature for it.
- POST */empty-sign*
  It builds a bump transaction and provides the signature for it.
- POST */send*
  It sends a bump or payment transaction for the transfer with the latest sequence.

Executive Summary

This report presents the findings of a code review and security analysis that was conducted between January 16, 2023 - February 09, 2023.

The purpose of this engagement was to evaluate the stability and security of the application against best practices and possible attack vectors.

The score is based only on the applications that were subject to “Code Review and Security Analysis” (see Scope).

For more details about the scoring, see Methodology.

Documentation quality

The technical documentation and functional requirements were sufficient.

The documentation quality score is **10** out of **10**.

Code quality

The code follows good practices, has a consistent style, and is generally easy to read. Tests cover the core functionality.

The code quality score is **10** out of **10**.

Architecture quality

It is not possible to increase the number of signatures that are required to transfer assets without changing the codebase. The current required amount is 2 signatures.

The architecture quality score is **7** out of **10**.

To improve the architecture quality score, design the architecture in a way where it is possible to add an arbitrary number of required signatures.

Security and Stability

As a result of the audit, security engineers found **2** critical and **1** low severity issues.

As a result of the remediation, **1** critical and **1** low severity issues were not fixed.

As a result of the second remediation, **1** low severity issue was not fixed.

A detailed description of the issues can be found in the Issue Overview section.

The security and stability score is **10** out of **10**.

Summary

According to the assessment, the Customer's applications have the following score: **9.7**

1
2
3
4
5
6
7
8
9
10
The final score

Checked Items

We have reviewed decentralized applications for commonly known and more specific vulnerabilities. Here are some of the items that were considered:

#	Item	Status
1	**Private keys are safely used and stored** Private keys should be encrypted, stored in a secure secret manager, provided to the application with a secure method.	Passed
2	**Strong authentication via message signing** A signed authentication message should be exclusive to the application. It should have an expiration date and time-specific nonce provided by the system.	Not Relevant
3	**Blockchain branching out is accounted for** The application should wait an adequate number of blocks to reduce the risk of a chain re-organization.	Passed
4	**Healthy integration with the blockchain nodes**	Passed
4.1	**Blockchain node redundancy** The application should not rely on a single blockchain data provider for a trusted and stable connectivity.	Passed
4.2	**Low-trust integration with the blockchain nodes** The application should expect that each interaction with a blockchain data provider can fail and handle these cases accordingly.	Passed
5	**Blockchain data is sanitized** The application should not trust the inputs from the blockchain when they can be considered the same as user inputs.	Passed
6	**Secure integration with the blockchain**	Passed
6.1	**Events are validated** The application should check the event origin, its type and structure.	Passed
6.2	**Proper mapping of types** Blockchain-specific types (uint256, bytes, etc.) should be mapped to the correct language-specific types without a data loss.	Passed
6.3	**Error-proof data indexing**	Passed

	The application should not silently skip anything it could not process at some point in time. It should have a retry mechanism or sufficient logging.	
6.4	**Secure integration with the smart contracts** The application should follow the smart contract specification and expect all relevant and possible outcomes.	Passed
7	**Dependencies are up-to-date and not vulnerable**	Passed

Severity Definitions

Risk Level	Description
Critical	Critical issues are usually straightforward to exploit, can lead to asset loss, data manipulations, or greatly impact the application's stability.
High	High-level issues are difficult to exploit. However, they also have a significant impact on the application execution.
Medium	Medium-level issues are important to fix. However, they cannot lead to asset loss or data manipulations.
Low	Low-level issues are mostly related to outdated, unused code snippets that cannot significantly impact execution.

Issue Overview

■■■■ Critical

C01. Exposure of Sensitive Information Through Environmental Variables

The applications use sensitive information such as private and API keys exposed to the environment. This practice can be easily abused and is considered unsafe from a security perspective.

stellar-bridge-validator/src/constants.ts

```
export const XLM_PRIVATE_KEY =
  process.env.XLM_PRIVATE_KEY &&
  CryptoJS.AES.decrypt(process.env.XLM_PRIVATE_KEY,
                                            AES_SECRET_KEY).toString(
    CryptoJS.enc.Utf8,
  );
```

bridge-stellar-server/src/constants.ts

```
export const PK = CryptoJS.AES.decrypt(process.env.PK,
                                            AES_SECRET_KEY).toString(
  CryptoJS.enc.Utf8,
);

export const NEAR_PK = CryptoJS.AES.decrypt(
  process.env.NEAR_PK,
  AES_SECRET_KEY,
).toString(CryptoJS.enc.Utf8);

export const QUEUE_API_KEY = process.env.QUEUE_API_KEY;

export const SECRET_KEY = process.env.SECRET_KEY;
export const ANOTHER_SERVER_SECRET_KEY =
                                  process.env.ANOTHER_SERVER_SECRET_KEY;
```

Although the private keys are AES encrypted, they are not considered securely encrypted because the encryption key is stored in the code as plain text.

stellar-bridge-validator/src/constants.ts

```
export const AES_SECRET_KEY = ' REDACTED ';
```

bridge-stellar-server/src/constants.ts

```
export const AES_SECRET_KEY = ' REDACTED ';
```

Recommendation: Do not expose sensitive information to the environment. Use the cloud provider's secret manager and its interface to access the secrets. See the cloud provider's documentation on how to work with secret values, as best practices may differ from one provider to another.

Status: Fixed (commit 5f992a3, 904078e)

C02. Replay Attack / DoS (bridge-stellar-server)

The endpoint POST */unlock* (method “unlock” of the “AppController” class) validates the provided signatures and sends a transaction to the NEAR blockchain creating a transfer. It is possible for an attacker to reuse valid old signatures to make the application send a transaction that will fail, but the bridge will still pay the fee for the transaction, thus making it possible to spend the entire balance on the NEAR blockchain and/or cause a DoS.

These 2 transactions are an example:

JCNZKP3eYqRdBS2JGMYkJgfuhBXpSe8QFFfJwHrfVQXY - it is a valid transaction that succeeded.

858h8uDKSi5869Hh22GZZAiQhqWDgrFDJvnDP9fZpXZ - this is a copy of a previous transaction that was sent by the application; although it failed, the transaction fee was paid.

See the bridge-stellar-server/src/app.controller.ts file, AppController.unlock method for more context.

Recommendation: Check if a transfer already exists. Do not allow reuse of old signatures.

Status: Fixed (commit 7e6a80a)

Low

L01. Claim Check Race Condition (stellar-bridge-validator)

The endpoint GET */sign/$TRANSACTION_ID* (method “getSignedMessage” of the “AppController” class) has a check that verifies that the transaction has not been claimed yet.

stellar-bridge-validator/src/app.controller.ts

```
await this.appService.checkIfClaimed(
  lockData.destination,
  source,
  lockData.lockId,
);
logger.info(`Not claimed`);
```

It uses a cached value or makes a request to the “bridge-stellar-server” application to perform this check. The nature of the asynchronous request / cache manager makes the race condition possible here.

Although this is an undesirable and unexpected behavior, this cannot affect the system in any meaningful way - the verification is redundant. The burden of verifying if the claim has been already made lies on the smart contract, thus making duplicate signatures irrelevant.

Recommendation: Remove the claim check or document in the code that this cannot reliably ensure that the claim has not been made already.

Status: Reported

Disclaimers

Hacken Disclaimer

The application given for the audit has been analyzed based on the best industry practices at the time of this report, in relation to cybersecurity vulnerabilities and issues in the application’s source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

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 contract statements. 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 the decentralized application.

Technical Disclaimer

Decentralized applications are closely integrated with a blockchain platform. The platform, its programming language, and other software related to the application can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited application.