InterBridge Relayer Audit Report

Fri Oct 25 2024

contact@bitslab.xyz https://twitter.com/tonbit_

TonBit

InterBridge Relayer Audit Report

1 Executive Summary

1.1 Project Information

Description	InterBridge is a liquidity pool-based bridge that allows users to add liquidity on Solana and TON. Users lock tokens on one chain, and InterBridge's infrastructure releases corresponding tokens to the users on the other chain.
Type	Bridge
Auditors	TonBit
Timeline	Fri Oct 11 2024 - Fri Oct 18 2024
Languages	Typescript
Platform	Ton,Solana
Methods	Dependency Check, Static Analysis, Manual Review
Source Code	https://github.com/soonlabs/cross-chain-bridge-relayer https://github.com/soonlabs/cross-chain-bridge-data-sync
Commits	https://github.com/soonlabs/cross-chain-bridge-relayer: bf2d541de61c02ac26e9d08ce4b4af5487429e88 ef26ecf8c7ff34707e120213ef8537836c09d10b https://github.com/soonlabs/cross-chain-bridge-data-sync: 357b7149d0834b1ffaa5437d939b9c3123efc7a4 afb16098c2745edfe3e1942a31d15110cd669c36

1.2 Files in Scope

The following are the directories of the original reviewed files.

Directory
https://github.com/soonlabs/cross-chain-bridge-relayer/src/transaction
https://github.com/soonlabs/cross-chain-bridge-data-sync/src/tx-sync

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	5	3
Informational	2	0	2
Minor	0	0	0
Medium	3	2	1
Major	2	2	0
Critical	1	1	0

1.4 TonBit Audit Breakdown

TonBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Integer overflow/underflow
- Infinite Loop
- Infinite Recursion
- Race Condition
- Traditional Web Vulnerabilities
- Memory Exhaustion Attack
- Disk Space Exhaustion Attack
- Side-channel Attack
- Denial of Service
- Replay Attacks
- Double-spending Attack
- Eclipse Attack
- Sybil Attack
- Eavesdropping Attack
- Business Logic Issues
- Contract Virtual Machine Vulnerabilities
- Coding Style Issues

1.5 Methodology

Our security team adopted **"Dependency Check"**, **"Automated Static Code Analysis"**, and **"Manual Review"** to conduct a comprehensive security test on the code in a manner closest to real attacks. The main entry points and scope of the security testing are specified in the **"Files in Scope"**, which can be expanded beyond the scope according to actual testing needs. The main types of this security audit include:

(1) Dependency Check

A comprehensive check of the software's dependency libraries was conducted to ensure all external libraries and frameworks are up-to-date and free of known security vulnerabilities.

(2) Automated Static Code Analysis

Static code analysis tools were used to find common programming errors, potential security vulnerabilities, and code patterns that do not conform to best practices.

(3) Manual Review

The scope of the code is explained in section 1.2.

(4) Audit Process

- Clarify the scope, objectives, and key requirements of the audit.
- Collect related materials such as software documentation, architecture diagrams, and lists of dependency libraries to provide background information for the audit.
- Use automated tools to generate a list of the software's dependency libraries and employ professional tools to scan these libraries for security vulnerabilities, identifying outdated or known vulnerable dependencies.
- Select and configure automated static analysis tools suitable for the project, perform automated scans to identify security vulnerabilities, non-standard coding, and potential risk points in the code. Evaluate the scanning results to determine which findings require further manual review.
- Design a series of fuzz testing cases aimed at testing the software's ability to handle exceptional data inputs. Analyze the issues found during the testing to determine the defects that need to be fixed.
- Based on the results of the preliminary automated analysis, develop a detailed code review plan, identifying the focus of the review. Experienced auditors perform line-by-

line reviews of key components and sensitive functionalities in the code.

- If any issues arise during the audit process, communicate with the code owner in a timely manner. The code owners should actively cooperate (this may include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- Necessary information during the audit process will be well documented in a timely manner for both the audit team and the code owner.

2 Summary

This report has been commissioned by InterSOON with the objective of identifying any potential issues and vulnerabilities within the source code of the InterBridge Relayer repository, as well as in the repository dependencies that are not part of an officially recognized library. In this audit, we have employed the following techniques to identify potential vulnerabilities and security issues:

(1) Dependency Check

A comprehensive analysis of the software's dependency libraries was conducted using the dependency analysis tool.

(2) Automated Static Code Analysis

The code quality was examined using a code scanner.

(4) Manual Code Review

The primary focus of the manual code review was:

- relayer
- data-sync

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
FOR-1	Solana Address Conversion Causes Cross-Chain Failures	Medium	Fixed
SSE-1	Potential Delay in `processTransactions` Function Due to Insufficient Liquidity	Medium	Acknowledged
SSE-2	Incorrect Use of DECIMALS May Cause Cross-Chain Failures for Users	Medium	Fixed

SSE-3	Relayer Should Verify Transaction Execution Status and On-Chain Status	Informational	Acknowledged
TSE-1	Incorrect Token Contract Address When Bridging from Solana to Ton May Cause Potential Asset Loss	Major	Fixed
TTS-1	A Single Ton or Solana Transaction May Contain Multiple Events	Major	Fixed
UTI-1	Insecure Authorization in `checkKey` Function Due to Weak Hashing Mechanism	Critical	Fixed
YAR-1	Dependency Known Vulnerability	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the InterBridge Relayer repository :

Ton Contract:

1. Users on Ton call the contract for cross-chain operations.
2. Admin sets the contract parameters.
3. Relayer calls the contract to transfer funds from Solana.

Solana Contract:

1. Users on Solana call the contract for cross-chain operations.
2. Admin sets the contract parameters.
3. Relayer calls the contract to transfer funds from Ton.

User: Calls the contract on either Ton or Solana for cross-chain operations.
Admin: Sets contract parameters, LP whitelist, etc.
Relayer: Receives on-chain events from Ton or Solana and performs cross-chain operations.
LP: Liquidity Provider.

4 Findings

FOR-1 Solana Address Conversion Causes Cross-Chain Failures

Severity: Medium

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/utils/format.ts#9

Descriptions:

1. When users transfer assets from Ton to Solana, a 64-byte address is provided. However, Solana addresses are 32 bytes, so the cross-chain bridge defaults to treating the upper 32 bytes of the 64-byte address as invalid (filled entirely with `0x00`).

2. Suppose a 64-byte cross-chain address is `0x00..00 + 0x00ff..ff`; the `bigintToSolanaAddr` function removes the first 32 bytes of `0x00` and also removes `0x00` from valid Solana addresses, resulting in an address with only 31 bytes.

```
export function bigintToSolanaAddr(addr: bigint) {
 return bs58.encode(Buffer.from(addr.toString(16), 'hex'));
}
```

3. In the `isValidSolanaAddress` function, if the address length is not 32 bytes, an error is returned.

```
export const isValidSolanaAddress = (address) => {
 try {
  const key = new PublicKey(address);
  return PublicKey.isOnCurve(key.toBytes());
 } catch (error) {
  return false;
 }
}
```

4. If a cross-chain target Solana address starts with `0x00`, the transfer will fail.

Suggestion:

Use a correct address conversion algorithm.

SSE-1 Potential Delay in `processTransactions` Function Due to Insufficient Liquidity

Severity: Medium

Discovery Methods: Manual Review

Status: Acknowledged

Code Location:

src/transaction/solana.service.ts#103-107;

src/transaction/ton.service.ts#145-148

Descriptions:

In the processTransactions function, there is a potential delay when processing transactions, particularly if the liquidity for a specific source token is insufficient. Below is the relevant code snippet:

```
    if (liquidityProviderBalance < SOLANA_MAXIMUM_TRANSFER_CAPACITY_PER_TX) {
      await sendSlackMessage('solana.service liquidity balance is less than ' +
SOLANA_MAXIMUM_TRANSFER_CAPACITY_PER_TX);

      return;
    };
```

Currently, the function processes one transaction at a time. If the liquidity for a specific sourceToken is insufficient (i.e., liquidityProviderBalance < SOLANA_MAXIMUM_TRANSFER_CAPACITY_PER_TX), the function will not proceed to the next transaction.

This design flaw can slow down transaction processing, as the function will repeatedly check the same transaction without advancing to others. This may degrade overall system performance and increase latency in transaction handling.

Suggestion:

To address this issue, it is advisable to implement a strategy where the function can process other transactions while retrying the transaction with insufficient liquidity after a certain period.

This could involve adding a mechanism to queue or defer the transaction for later processing, thus ensuring that the processTransactions flow remains efficient.

Resolution:

The next version will handle multiple transactions simultaneously, which will address this issue.

SSE-2 Incorrect Use of DECIMALS May Cause Cross-Chain Failures for Users

Severity: Medium

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/transaction/solana.service.ts#283;

src/transaction/ton.service.ts#298

Descriptions:

When the relayer calculates whether the cross-chain amount exceeds the "SOLANA_MAXIMUM_TRANSFER_CAPACITY_PER_TX" or "TON_MAXIMUM_TRANSFER_CAPACITY_PER_TX," it uses the decimal of USDT. This may lead to cross-chain failures for users.

```
 async sendTransaction(guid: string, userAddress: string, tokenAmount: string) {
   try {
     if (Number(tokenAmount) / (10 **
this.envConfig.SOLANA_SUPPORTED_TOKENS.USDT.DECIMALS) >
SOLANA_MAXIMUM_TRANSFER_CAPACITY_PER_TX) {
       return;
     }
```

Suggestion:

Calculate according to the decimals of different tokens

SSE-3 Relayer Should Verify Transaction Execution Status and On-Chain Status

Severity: Informational

Discovery Methods:

Status: Acknowledged

Code Location:

src/transaction/solana.service.ts#366;

src/transaction/ton.service.ts#311

Descriptions:

When sending transactions on Solana or Ton, the relayer should check whether the transaction has executed successfully and whether the block containing the transaction has been finalized.

Suggestion:

1. Add a retry mechanism for failed transactions.
2. Confirm whether the block containing the transaction may be rolled back.

Resolution:

The relayer will execute the transaction only after waiting 1 minute for block confirmation.

TSE-1 Incorrect Token Contract Address When Bridging from Solana to Ton May Cause Potential Asset Loss

Severity: Major

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/transaction/ton.service.ts#316

Descriptions:

1. When bridging from Solana to Ton, the relayer needs to correctly identify the token type for the cross-chain transaction. However, in the `sendTransaction` function, all token types are treated as `USDT` regardless of the actual type.
2. If there is a token X on Solana that is cheaper than USDT, an attacker could exploit this by exchanging the cheaper token X for the more expensive USDT.

```
 async sendTransaction(guid: string, userAddress: string, tokenAmount: string) {
   try {
     //
//......other code......
//
     // jetton minter contract
     const jettonMinterContract =
JettonMinter.createFromAddress(Address.parse(this.envConfig.TON_SUPPORTED_TOKENS.
     // bridge gate contract
     const bridgeGate =
this.client.open(BridgeGate.createFromAddress(Address.parse(this.envConfig.TON_BRIDGE

     const bridgeAmount = BigInt(tokenAmount);
     //
//......other code......
//
```

3. Currently supported cross-chain tokens include native SOL and TON tokens. The following is the configuration file showing other supported token types.

```
TON_SUPPORTED_TOKENS: {
  USDT: "",
  USDC: ""
},
TON_BRIDGE_GATE: "UQCYGovtmogjl_nVFLfu2QouyNLbwFQ9wxqorZUU6UTe5xND",
SOLANA_RPC_URL: "",
SOLANA_SUPPORTED_TOKENS: {
  USDT: "",
  USDC: ""
},
```

Suggestion:

Correct the token contract address.

TTS-1 A Single Ton or Solana Transaction May Contain Multiple Events

Severity: Major

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/tx-sync/chains/ton.tx.sync.service.ts#173;

src/tx-sync/chains/solana.tx.sync.service.ts#72

Descriptions:

1. The data-sync service, when retrieving cross-chain events, fetches the first cross-chain event in the transaction.
2. A transaction may contain multiple cross-chain events. For example, a Solana transaction is composed of multiple instructions, and each instruction can call the `"bridge_to_destination"` function, leading to multiple cross-chain events within a single transaction.
3. This situation may cause cross-chain failures for users.
4. `getSignaturesForAddress` could get all txs related to ProgramId. If the attacker sends similar logs, the previous code may cause asset loss.

Suggestion:

1. In general, users should not construct multiple instructions within a single transaction.
2. It is recommended to include a warning in the project development documentation or user manual, advising users against constructing multiple cross-chain operations within a single transaction.
3. It is recommended to use another way to get all txs.

UTI-1 Insecure Authorization in `checkKey` Function Due to Weak Hashing Mechanism

Severity: Critical

Discovery Methods: Manual Review

Status: Fixed

Code Location:

src/utils.ts#11-18

Descriptions:

The `checkKey` function uses an insecure authorization mechanism that relies on a weak hashing function `hashCode`. Below is the code in question:

```
export function checkKey(key: string) {
  if (key == undefined) return false;
  const decrypt = hashCode(key);
  if (decrypt != ADMIN_KEY_HASH) {
    return false;
  }
  return true;
}

export const hashCode = (s) => {
  return s.split('').reduce(function (a, b) {
    a = (a << 5) - a + b.charCodeAt(0);
    return a & a;
  }, 0);
};
```

The `hashCode` function is not a cryptographic hash function and is highly vulnerable to collisions. It only performs very simple bitwise and addition operations.

Moreover, the function's input length and output length follow a clear and predictable pattern, making it easy to reverse and crack.

Due to this weak hashing mechanism, attackers can easily generate keys that result in the same `ADMIN_KEY_HASH` value, which compromises the security of the API's authorization process.

The proof-of-concept (PoC) code demonstrating how the weak hashing can be exploited is as follows:

```
const generateRandomString = (length: number): string => {
  const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  let result = '9';
  const charactersLength = characters.length;
  for (let i = 1; i < length; i++) {
    result += characters.charAt(Math.floor(Math.random() * charactersLength));
  }
  return result;
};

const poc = () => {
  const attempts = 10000000000;
  for (let i = 0; i < attempts; i++) {
    const randomString = generateRandomString(5);
    if (checkKey(randomString)) {
      console.log(`Found valid key: ${randomString}`);
    }
  }
};

poc();
```

Suggestion:

It is highly recommended to replace the `hashCode` function with a secure cryptographic hash function (e.g., SHA-256) to strengthen the hashing process.

Additionally, implementing proper key management and secure authorization practices will ensure a more robust and secure system.

YAR-1 Dependency Known Vulnerability

Severity: Informational

Discovery Methods: Dependency Check

Status: Acknowledged

Code Location:

yarn.lock#1

Descriptions:

The project contains known vulnerabilities in two transitive dependencies:

path-to-regexp: outputs backtracking regular expressions. More Info

body-parser: Vulnerable to denial of service (DoS) attacks when URL encoding is enabled. More Info

Suggestion:

It is recommended to update these dependencies to mitigate potential risks associated with the known vulnerabilities.

Resolution:

Plans to address this issue in the next version.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.