SALUS SECURITY

JUN 2024

CODE SECURITY ASSESSMENT

LORENZO PROTOCOL

Overview

Project Summary

- Name: Lorenzo Protocol - stBTC bridge
- Platform: Lorenzo and Bitlayer
- Language: Solidity
- Repository:
    - https://github.com/Lorenzo-Protocol/stBTC-bridge
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Lorenzo Protocol - stBTC bridge
Version	v2
Type	Solidity
Dates	Jun 07 2024
Logs	Jun 06 2024; Jun 07 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	1
Total Low-Severity issues	0
Total informational issues	2
Total	4

Contact

E-mail: support@salusec.io

SALUS

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

SALUS

Content

SALUS

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

SALUS

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Should burn stBTC when users convert stBTC to BTC	High	Business Logic	Resolved
2	Centralization Risk	Medium	Centralization	Acknowledged
3	Gas optimization suggestions	Informational	Gas Optimization	Resolved
4	Events are not indexed	Informational	Logging	Resolved

SALUS

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Should burn stBTC when users convert stBTC to BTC

Severity: High	Category: Business Logic
Target: - contracts/BridgeV2.sol	

Description

In BridgeV2, there are functions to convert tokens between BTC and stBTC. When users want to convert BTC to stBTC, they need to send BTC to the contract and the contract will mint a related amount of stBTC tokens for the user. When users want to convert stBTC to BTC, the contract should burn stBTC tokens and transfer BTC back to the users.

The vulnerability is that the contract does not burn stBTC tokens when users want to convert stBTC to BTC. In this case, users can get back their BTC, while holding stBTC tokens.

contracts/BridgeV2.sol:L272-L289

```
function convertstBTC2BTC(uint256 amount) external {
    if (block.timestamp < _withdrawalTimestamp) {
        revert WithdrawalTimeNotReach();
    }
    if (_stakerInfo[msg.sender] < amount) {
        revert InvalidAmount();
    }
    if (address(this).balance < amount) {
        revert NotEnoughEthToWithdrawal();
    }
    _stakerInfo[msg.sender] -= amount;
    (bool success, ) = payable(msg.sender).call{value: amount}("");
    if (!success) {
        revert SendETHFailed();
    }

    emit ConvertstBTC2BTC(msg.sender, amount);
}
```

Recommendation

When users convert stBTC back to BTC, burn the related stBTC tokens.

Status

This issue has been resolved by the team with commit 38f1d5d.

SALUS

2. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/BridgeV2.sol	

Description

There is a privileged owner role in the BridgeV2 contract. The owner of the BridgeV2 contract can withdraw all BTC in the smart contract.

Should the owner's private key be compromised, an attacker could withdraw all BTC.

contracts/BridgeV2.sol:L291-L306

```
function withdrawBTC(uint256 amount, address to) external onlyOwner {
    if (to == address(0x0)) {
        revert InvalidReceiver();
    }
    if (block.timestamp > _withdrawalTimestamp) {
        revert CannotWithdrawal();
    }
    if (address(this).balance < amount) {
        revert NotEnoughEthToWithdrawal();
    }
    (bool success, ) = payable(to).call{value: amount}("");
    if (!success) {
        revert SendETHFailed();
    }
    emit WithdrawBTC(msg.sender, to, amount);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

SALUS

2.3 Informational Findings

3. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - contracts/BridgeV2.sol	

Description

Finding 1: When a user wants to bridge stBTC(bitlayer ERC20 token) back to Lorenzo, the user needs to approve the amount to bridge contract address and then call the burn0rStakeStBtc() function, which will burn stBTC(erc20) token.

contracts/BridgeV2.sol:L173

```
function burnOrStakeStBtc(
      uint256 amount,
      uint256 toChainId,
      address receiver) external payable whenNotPaused {
      ...
      IERC20MintBurnable(stBTCAddress).burnFrom(msg.sender, amount);
      ...
}
```

If there is a `burn()` function similar to the mint() function on stBTC smart contract, the user can skip the approval call which can save the gas.

Finding 2:

contracts/BridgeV2.sol:L160-L165

```
function burnOrStakeStBtc(...){
    ...
    address stBTCAddress = _supportChain[_chainId].stBTCAddress;
    ...
    if (_supportChain[_chainId].stBTCAddress == NATIVE_TOKEN) {
    ...
}
```

contracts/BridgeV2.sol:L225

```
function mintOrUnstakeStBtcByArray(...){
    ...
    for (uint256 i = 0; i < to.length; i++){
    ...
    address stBTCAddress = _supportChain[_chainId].stBTCAddress;
    ...
}
```

Memory reading saves more gas than storage reading multiple times when the state is not changed. So caching the storage variables in memory and using the memory instead of storage reading is effective.

SALUS

Recommendation

Use suggestions to save gas.

Status

This issue has been resolved by the team with commit ce04518.

SALUS

4. Events are not indexed

Severity: Informational	Category: Logging
Target: - contracts/BridgeV2.sol	

Description

Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes that allow users to evaluate them. The emitted events are not indexed, making off-chain scripts such as front-ends of dApps difficult to filter the events efficiently.

In the BridgeV2 contract, the `Mint` and `Burn` events are not indexed.

Recommendation

Consider adding the indexed keyword in the `Mint` and `Burn` events.

Status

This issue has been resolved by the team with commit ce04518.

SALUS

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 220fe73:

File	SHA-1 hash
contracts/BridgeV2.sol	e8349c6b3e7390a994ad2fcc9766491f99c3e4a6
contracts/storage/BridgeStorageV2.sol	cf1b3590bd09c3e6a9e0291644b0c4501223f427

SALUS