SALUS SECURITY

MAR 2024

# CODE SECURITY ASSESSMENT

BITMAP TECH

# Overview

## Project Summary

- Name: Bitmap Tech - BTCBridge
- Platform: Merlin Chain
- Language: Solidity
- Repository: https://github.com/MerlinLayer2/BTCLayer2BridgeContract
- Audit Scope: See Appendix - 1

# Project Dashboard

## Application Summary

| | |
|---|---|
| **Name** | Bitmap Tech - BTCBridge |
| **Version** | v3 |
| **Type** | Solidity |
| **Date** | Mar 06 2024 |
| **Logs** | Feb 20 2024; Mar 04 2024; Mar 06 2024 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 1 |
| **Total Low-Severity issues** | 4 |
| **Total informational issues** | 4 |
| **Total** | 9 |

## Contact

E-mail: support@salusec.io

# 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. |

# Content

# 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.

# Findings

## 2.1 Summary of Findings

| ID | Title | Severity | Category | Status |
|---|---|---|---|---|
| 1 | Missing update unlockTokenAdminAddressList when delete admin | Medium | Business Logic | Resolved |
| 2 | The tokenURI cannot be queried by the public | Low | Access Control | Resolved |
| 3 | The inscriptionId can be overridden | Low | Business Logic | Resolved |
| 4 | Contracts cannot be deployed correctly | Low | Business Logic | Acknowledged |
| 5 | Centralization risk | Low | Centralization | Acknowledged |
| 6 | Lack of upper limit for fees in Lack of upper limit for fees in setBridgeSettingsFee()() | Informational | Centralization | Resolved |
| 7 | Implementation can be initialized | Informational | Configuration | Acknowledged |
| 8 | Lack of indexed parameters in events | Informational | Logging | Acknowledged |
| 9 | Missing zero address check | Informational | Data Validation | Acknowledged |

## 2.2 Notable Findings

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

### 1. Missing update unlockTokenAdminAddressList when delete admin

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

### Description

In BTCLayer2Bridge.sol, the functions addUnlockTokenAdminAddress() and delUnlockTokenAdminAddress() are used to add and remove admin addresses respectively.

contracts/BTCLayer2Bridge.sol:L153-L166

```
function addUnlockTokenAdminAddress(address _account) public onlyValidAddress(_account)
{
    require(msg.sender == superAdminAddress || msg.sender == normalAdminAddress,
"Illegal permissions");
    require(unlockTokenAdminAddressSupported[_account] == false, "Current address has
been added");
    unlockTokenAdminAddressList.push(_account);
    unlockTokenAdminAddressSupported[_account] = true;
    emit AddUnlockTokenAdminAddress(_account);
}

function delUnlockTokenAdminAddress(address _account) public onlyValidAddress(_account)
{
    require(msg.sender == superAdminAddress || msg.sender == normalAdminAddress,
"Illegal permissions");
    require(unlockTokenAdminAddressSupported[_account] == true, "Current address is not
exist");
    unlockTokenAdminAddressSupported[_account] = false;
    emit DelUnlockTokenAdminAddress(_account);
}
```

However, the delUnlockTokenAdminAddress() function is missing the step to remove the address from the unlockTokenAdminAddressList array. This results in the deleted address remaining in the admin array.

### Recommendation

Consider removing the address from the unlockTokenAdminAddressList in the delUnlockTokenAdminAddress() function.

### Status

The team has resolved this issue in commit 30be513a.

## 2. The tokenURI cannot be queried by the public

| Severity: Low | Category: Access Control |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

## Description

The tokenURI() function should be designed as a public query function to allow users easy access to critical information, thereby improving the user interaction experience.

contracts/BTCLayer2Bridge.sol:L204-L207

```
function tokenURI(address token, uint256 inscriptionNumber) public returns (string
memory) {
    require(msg.sender == superAdminAddress || msg.sender == normalAdminAddress,
"Illegal permissions");
    return IBTCLayer2BridgeERC721(bridgeERC721Address).tokenURI(token,
inscriptionNumber);
}
```

However, the current code incorrectly enforces access control on the tokenURI() function, preventing users from querying this essential information.

## Recommendation

Consider removing the highlighted part of the code above and defining the function as a view function.

## Status

The team has resolved this issue in commit a0cbbb08.

## 3. The inscriptionId can be overridden

| Severity: Low | Category: Business Logic |
| --- | --- |
| Target:<br>- contracts/ERC721TokenWrapped.sol | |

## Description

In ERC721TokenWrapped.sol, there should be a one-to-one correspondence between inscriptionId and inscriptionNumber.

contracts/ERC721TokenWrapped.sol:L32-L37

```
function mint(address to, uint256 inscriptionNumber, string memory inscriptionId)
external onlyBridge {
    mpId2Number[inscriptionId] = inscriptionNumber;
    mpNumber2Id[inscriptionNumber] = inscriptionId;

    _mint(to, inscriptionNumber);
}
```

However, the mint() function lacks a validation to check if the inscriptionId already exists. This could lead to an existing inscriptionId being overwritten by another inscriptionNumber, resulting in an anomaly in the correspondence between inscriptionId and inscriptionNumber.

contracts/ERC721TokenWrapped.sol:L63-L66

```
function tokenURI(uint256 inscriptionNumber) public view override virtual returns
(string memory) {
    string memory inscriptionId = mpNumber2Id[inscriptionNumber];
    return bytes(_baseTokenURI).length > 0 ? string.concat(_baseTokenURI, inscriptionId)
: "";
}
```

A deeper implication is that two different tokenId's can correspond to the same tokenURI.

## Recommendation

It is recommended to check if the inscriptionId is being used in the mint() function.
For example, the mint() function can add the following judgment:

```
function mint(address to, uint256 inscriptionNumber, string memory inscriptionId)
external onlyBridge {
    require(mpId2Number[inscriptionId] == 0, "Invalid inscriptionId");
    mpId2Number[inscriptionId] = inscriptionNumber;
    mpNumber2Id[inscriptionNumber] = inscriptionId;

    _mint(to, inscriptionNumber);
}
```

## Status

The team has resolved this issue in commit 313285d9.

## 4. Contracts cannot be deployed correctly

| Severity: Low | Category: Business Logic |
| --- | --- |

Target:
- contracts/BTCLayer2Bridge.sol
- contracts/BTCLayer2BridgeERC20.sol
- contracts/BTCLayer2BridgeERC721.sol

## Description

contracts/BTCLayer2Bridge.sol:L121-L138

```
function initialize(
    ...
    address _bridgeERC20Address,
) external onlyValidAddress(_bridgeERC20Address)
virtual initializer {
    ...
    bridgeERC20Address = _bridgeERC20Address;
    ...
}
```

In BTCLayer2Bridge.initialize(), the address of the BTCLayer2BridgeERC20 contract needs to be provided as an initialization parameter.

contracts/BTCLayer2BridgeERC20.sol:L32-L40

```
function initialize(
    address _initialOwner,
    address _bridgeAddress
) external onlyValidAddress(_initialOwner)
onlyValidAddress(_bridgeAddress) virtual initializer {
    bridgeAddress = _bridgeAddress;
    // Initialize OZ contracts
    __Ownable_init_unchained(_initialOwner);
}
```

However, in BTCLayer2BridgeERC20.initialize(), the address of the BTCLayer2Bridge contract also needs to be provided as an initialization parameter.

However, according to the ERC1967 specification, the deployment of the proxy contract and the initialization of the Implementation should be executed in the same transaction. Therefore this interdependency results in the inability to properly initialize both proxy contracts.

The same issue exists between BTCLayer2Bridge and BTCLayer2BridgeERC721.

## Recommendation

Consider setting the address of the bridge outside the initialize() function

## Status

This issue has been acknowledged by the team.

## 5. Centralization risk

| Severity: Low | Category: Centralization |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

## Description

In the BTCLayer2Bridge contract, there is a privileged unlockTokenAdmin role. The unlockTokenAdmin role has the ability to:

- add/delete UnlockTokenAdminAddress
- mint ERC20 token
- unlock native token

If the unlockTokenAdmin’s private key is compromised, the attacker can exploit the admin's role to extract any amount of native token in the contract. In this scenario, users' assets would be In this scenario, users' assets would be severely compromised.

## Recommendation

We recommend transferring the admin role to a multisig account with a timelock feature for enhanced security. This ensures that no single person has full control over the account and that any changes must be authorized by multiple parties.

## Status

This issue has been acknowledged by the team.

## 2.3 Informational Findings

### 6. Lack of upper limit for fees in setBridgeSettingsFee()

| Severity: Informational | Category: Centralization |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

### Description

The setBridgeSettingsFee() function is used to modify the value of bridgeFee.

contracts/BTCLayer2Bridge.sol:L287-L298

```
function setBridgeSettingsFee(address _feeAddress, uint256 _bridgeFee) external {
    require(msg.sender == superAdminAddress, "Illegal permissions");

    if (_feeAddress != address(0)) {
        feeAddress = _feeAddress;
    }
    if (_bridgeFee > 0) {
        bridgeFee = _bridgeFee;
    }

    emit SetBridgeSettingsFee(_feeAddress, _bridgeFee);
}
```

However, the function has no upper limit on fees. If the fee setter's private key is compromised, the attacker can set bridgeFee to type(uint256 ).max, preventing users from performing cross-chain operations.

### Recommendation

It is recommended to include a reasonable upper limit for bridgeFee in setBridgeSettingsFee().

### Status

The team has resolved this issue in commit 8967db4c.

## 7. Implementation can be initialized

| Severity: Informational | Category: Configuration |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

## Description

The BTCLayer2Bridge contract uses the Initializable module from OpenZeppelin. According to OpenZeppelin's documentation, it's best to invoke the `_disableInitializers` function in the constructor to prevent the implementation contract from being initialized by malicious users.

## Recommendation

Consider using the _disableInitializers() function in the constructor to prevent malicious initialization of the Implement contract.

## Status

This issue has been acknowledged by the team.

## 8. Lack of indexed parameters in events

| Severity: Informational | Category: Logging |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

### Description

Event emission and properly indexed parameters assist off-chain observers watch, search, and filter on-chain activity.

In BTCLayer2Bridge.sol, many key events are missing the indexed parameters.

- MintERC20Token
- BurnERC20Token
- UnlockNativeToken
- LockNativeToken

### Recommendation

Consider indexing applicable event parameters to support the searching and filtering abilities of offchain services.

### Status

This issue has been acknowledged by the team.

## 9. Missing zero address check

| Severity: Informational | Category: Data Validation |
| --- | --- |
| Target:<br>- contracts/BTCLayer2Bridge.sol | |

## Description

contracts/BTCLayer2Bridge.sol:L244-L250

```
function lockNativeToken(string memory destBtcAddr) public payable {
    require(msg.value > bridgeFee, "Insufficient cross-chain assets");

    (bool success, ) = feeAddress.call{value: bridgeFee}(new bytes(0));
    if (!success) {
        revert EtherTransferFailed();
    }

    emit LockNativeToken(msg.sender, msg.value - bridgeFee, destBtcAddr);
}
```

The lockNativeToken() function is used to cross-chain to BTC.

In this case, if the user incorrectly passes a zero address as a parameter, it will result in an off-chain processing exception.

## Recommendation

It is recommended to perform a zero address check on the parameter destBtcAddr in the lockNativeToken() function.

## Status

This issue has been acknowledged by the team.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit 20a8718:

| File | SHA-1 hash |
|---|---|
| BTCLayer2Bridge.sol | e1674de869992420fa2f3eb369883107405e710e |
| BTCLayer2BridgeERC20.sol | 790580c9eb981317e6e640640a058da6e11668f3 |
| BTCLayer2BridgeERC721.sol | f62299f91a29998ed8eb7268e36f6817faa8dbfd |
| ERC20TokenWrapped.sol | 6641f16b20952a42b963fb232803143d8d5b953a |
| ERC721TokenWrapped.sol | 5e7089e5d68f67289f5c050de5311a96e215837a |