AILayer Smart Contract
Audit Report

Fri May 17 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

AILayer Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	An asset cross-chain project
Type	DeFi
Auditors	ScaleBit
Timeline	Thu Mar 28 2024 - Thu Mar 28 2024
Languages	Solidity
Platform	AILayer
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/AINNLayer2/anvm_bridge
Commits	126abe83f0ebae16abcf9acd58481fe37398adcd

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BTCB	contracts/btc-bridge/BTCBridge.sol	4b01e86c8d321acdf5b6ca3b932ddbc117c17653
IBTCB	contracts/btc-bridge/interfaces/IBTCBridge.sol	cf31b31521163837abd0cd2f75c0b0fd25097037
IBTCV	contracts/btc-bridge/interfaces/IBTCVault.sol	91cad9a90c27879662b70cf7a74ca96a3e1723aa
BTCV	contracts/btc-bridge/BTCVault.sol	a260d5d7d384d6b77bb4fef99a6d3b12ab55a40d
BTCBB	contracts/btc-bridge/BTCBridgeBase.sol	b08383fa4406da36a399cdd052787109eb670601

1.3 Issue Statistic

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

1.4 ScaleBit Audit Breakdown

ScaleBit 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):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by AILayer to identify any potential issues and vulnerabilities in the source code of the AILayer smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 2 issues of varying severity, listed below.

ID	Title	Severity	Status
BTC-1	Centralization Risk	Major	Acknowledged
BTC-2	Missing 0 Address Check	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the AILayer Smart Contract :

Owner

- The Owner can set the `minWithdrawAmount` through `setMinWithdrawAmount()` .
- The Owner can set the `withdrawFee` through `setWithdrawFee()` .
- The Owner can set the `depositFee` through `setDepositFee()` .
- The Owner can set the `feeRecipient` through `setFeeRecipient()` .
- The Owner can add an active bridge through `addBridge()` .
- The Owner can set the bridge state to inactive through `deactivateBridge()` .
- The Owner can set the bridge state to active through `activateBridge()` .

Anchor

- The Anchor can send `ETH` to the `recipientEVM` address of the user through `receiveFromAnchor()` .

User

- The USer can send `Eth` to the vault through `withdraw()` .

4 Findings

BTC-1 Centralization Risk

Severity: Major

Status: Acknowledged

Code Location:

contracts/btc-bridge/BTCVault.sol#70,76;

contracts/btc-bridge/BTCBridge.sol#144

Descriptions:

Centralization risk was identified in the smart contract.

- The owner can change the bridge state through `deactivateBridge()` and `activateBridge()`.
- The owner can set the `minWithdrawAmount` through `setMinWithdrawAmount()`.

Suggestion:

It is recommended to take measures to mitigate this issue.

BTC-2 Missing 0 Address Check

Severity: Informational

Status: Acknowledged

Code Location:

contracts/btc-bridge/BTCVault.sol#35-50,159

Descriptions:

Missing 0 address check, the code is not rigorous.

```
    constructor(
    IBTCVault vault_,
    address anchor_,
    address initialOwner_,
    address initialModerator_
)
    BoolConsumerBaseV2(anchor_)
    BTCBridgeBase(uint256(10 ** (vault_.l2Decimals() - vault_.l1Decimals())))
    BridgePausable(initialModerator_)
{
    vault = vault_;
    bitcoinChainId = vault_.bitcoinChainId();
    messenger = IAnchor(anchor_).messenger();
    _transferOwnership(initialOwner_);
    feeRecipient = initialOwner_;
}

function setFeeRecipient(address newRecipient) external onlyOwner {
    feeRecipient = newRecipient;
    emit SetFeeRecipient(newRecipient);
}
```

Suggestion:

It is recommended to add 0 address check.

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.