Lumoz

Audit Report

Thu Nov 28 2024

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

ScaleBit

Lumoz Audit Report

1 Executive Summary

1.1 Project Information

Description	The Lumoz Protocol, as a globally distributed modular computing protocol, offering a powerful, secure, and flexible computing platform for users worldwide
Type	Infra
Auditors	ScaleBit
Timeline	Tue Nov 19 2024 - Thu Nov 28 2024
Languages	Solidity
Platform	Arbitrum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Lumoz-protocol/Lumoz-protocol-contracts https://github.com/Lumoz-protocol/Lumoz-zkProver-contracts https://github.com/Lumoz-protocol/Lumoz-zkVerifier-contracts
Commits	https://github.com/Lumoz-protocol/Lumoz-protocol-contracts: 4dcbed11578e541d01983df674b6cd3c987e481f 636be6d26c68d042119ab7c4c676e540afb56a25 https://github.com/Lumoz-protocol/Lumoz-zkProver-contracts: f4a9a8e95a0a48c93f0ffb8590a7a5b961e3197c

	https://github.com/Lumoz-protocol/Lumoz-zkVerifier-contracts: 4680f9ef1621c3f099dc484cc7bfcca4c6158020 ca304d3513a64deebc50bf36a85d92c264531fe6 5a65a761c9c45d6405d46c8b42128a912124f444

1.2 Files in Scope

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

ID	File	SHA-1 Hash
LOGCC	contracts/LumozOGClaimContract.sol	a46c6589ad56541a16bde19f1fa9c88b94631fd6
LTA	contracts/LumozTokenAirdrop.sol	44ce927cf1b8575e10eced502e1e295d2bf89c2a
LOGA	contracts/LumozOGAirdrop.sol	28ebac98d2e9411a658e9295e0499cf5b49bbca6
UTI	contracts/util.sol	3dc2cd4c3080da8b3af57dc59ef4009d89c9c1fb
LOGCC1	contracts/LumozOGConvertContract.sol	7c2518b08d501e5fb21b9c9e499b4fe332dada45
LOGNFT	contracts/LumozOGNFT.sol	4968b0e763697df8fd8ea128dc317b875fb215c2
LBA	contracts/LumozBaseAirdrop.sol	2a88912863b8b7ae45f69652b60241df6f3b1c1f
ZKPP	contracts/ZKProverProtocol.sol	80da513ea47e7cf0e3d2a60a145566e42615a7ff
EMOZ	contracts/esMOZ.sol	e2c04dddafa0b6816c7f923645c45b0693eecab5
ZVM	contracts/node_manager/zkVerifiersMulticall.sol	6fdb2fcbb84436ed2a390a9d5659c7181521e087
NMA	contracts/node_manager/NodeManager.sol	e8414971cdbe0b992e79731455dbc1ecd8e96588

STR	contracts/node_manager/StakedTracker.sol	c7d91d2486f9641a6d7017092e0012f36b200116
NFA	contracts/node_manager/NodeFactory.sol	0db8867c30f3946f8ac9ae8042f933768f559115
NPD	contracts/node_manager/NodeProxyDeployer.sol	766005bd3f56bf5aff1ae23cbe764bc23e582cd1
MAT	contracts/node_manager/math.sol	fb4cc0e4e9d2c3084abe70e712d3c37384e8aa38
NBE	contracts/node_manager/NodeBeacon.sol	31b4816630d250f78f80bd9eeffafa2b6f3a61d9
ZKVNP	contracts/ZKVerifierNodeProtocol.sol	aac13c78ea6a2ed71c1bc31f8dd59e89db2e220d
ILOGNFT	contracts/interfaces/ILumozOGNFT.sol	4004e1d192db6a900f46e2d3683d962a999ad850
ITA	contracts/interfaces/ITask.sol	b8f3ec4f18faf582c2f94e52bb31507e98a3fd36
LCC	contracts/LicenseClaimContract.sol	831ce10f0192569616f679d858302df81b553797
TMA	contracts/task/TaskManager.sol	fe4ffed34af0c514fa73130bbe3f8db3e9a07f2f
NTA	contracts/task/NormalTask.sol	0f1acb144d9772c8aee1e0f78dcaba9fa3acb549
NLI	contracts/NodeLicense.sol	673d51aaae84ea760ca03504b2bf3acdfaad7cdc

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	10	10	0
Informational	2	2	0
Minor	2	2	0
Medium	4	4	0
Major	2	2	0
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 Lumoz to identify any potential issues and vulnerabilities in the source code of the Lumoz 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 10 issues of varying severity, listed below.

ID	Title	Severity	Status
EMO-1	Incorrect Usage of `_pauseConvertToMOZ` in `convertToMOZ` Function	Medium	Fixed
EMO-2	Potential Array Out-of-Bounds Access	Minor	Fixed
LBA-1	Single-step Ownership Transfer Can be Dangerous	Medium	Fixed
LCC-1	Signature Replay Attack	Major	Fixed
LOG-1	Missing `disableInitializers` Call in Proxy Upgradeable Contract Constructor	Medium	Fixed
LOG-2	Logical Flaw in Blacklist Check for `LumozOGNFT` Contract	Medium	Fixed
LTA-1	Use `safeTransfer()` instead of `transfer()`	Major	Fixed
LTA-2	Redundant Address Validation in claims Function	Informational	Fixed

NFA-1	Missing Return Value Check	Minor	Fixed
LOG1-1	Incorrect Order of Validation Conditions in `claim` Function	Informational	Fixed

3 Participant Process

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

- `setProposalAuthority` : Transfers ProposalAuthority to another address.
- `setReviewAuthority` : Transfers ReviewAuthority to another address.
- `changeStatus` : Enables or disables a specific contract functionality.
- `updateVerifier` : Updates the verifier's address.
- `updateVerifierRole` : Grants or revokes the verifier role for a specific address.
- `updateMaxClaimedCount` : Updates the maximum tokens claimable per transaction.
- `updateMaxBurnCount` : Updates the maximum tokens burnable per transaction.
- `updateTokenAddress` : Updates the token contract address.
- `updateNodeLicenseAddress` : Updates the Node License contract address.
- `updateSignatureInterval` : Updates the validity interval for signatures.
- `setBaseURI` : Sets the base URI for NFTs.
- `pause` : Pauses the contract.
- `unpause` : Resumes the contract.
- `enableNode` : Enables node-related functionality.
- `changeUpdateSharesPendingPeriod` : Updates the delay period for pending share updates.
- `initShares` : Initializes the share distribution.
- `updateShares` : Updates the share distribution configuration.
- `updateMetadata` : Updates metadata and social details.
- `updateDelegateOwner` : Updates the delegate owner.
- `addToWhitelist` : Adds an address to the whitelist.
- `removeFromWhitelist` : Removes an address from the whitelist.

- `changeRedemptionStatus`: Enables or disables the redemption process.
- `changeConvertToMOZStatus`: Enables or disables the conversion from `esMOZ` to `MOZ`.
- `updateFoundationBasePoints`: Updates the foundation's base points for `esMOZ`.
- `setOGAddress`: Sets the OG token contract address.
- `closeContract`: Closes the contract.
- `setRewardPerVerifytask`: Updates the reward per verification task.
- `updateMaxStakeEsMOZ`: Updates the maximum stakable amount of `esMOZ`.
- `updateMaxStakeLicenseNum`: Updates the maximum stakable number of Licenses.
- `updateStakingTier`: Updates the threshold and boost factor of a staking tier.
- `addStakingTier`: Adds a new staking tier.
- `removeStakingTier`: Removes an existing staking tier.
- `safeMint`: Safely mints a token to a specified address.
- `mint`: Mints a token to a specified address.
- `batchMint`: Mints multiple tokens to multiple addresses.
- `burn`: Burns a specified token.

User

- `claim`: Claims rewards or tokens using provided data and signatures.
- `convert`: Converts tokens by burning them.
- `createSingleNode`: Creates a node with specified licenses, shares, and metadata.
- `stakeLicenses`: Stakes Licenses.
- `unstakeLicenses`: Unstakes Licenses.
- `stakeEsMOZ`: Stakes `esMOZ` tokens.
- `unstakeEsMOZ`: Unstakes `esMOZ` tokens.
- `stakeMOZ`: Converts `MOZ` tokens to `esMOZ` and stakes them.
- `claim`: Claims rewards for a user.

- `claimForOwner`: Claims rewards on behalf of the owner.
- `startRedemption`: Starts the `esMOZ` redemption process.
- `cancelRedemption`: Cancels an ongoing redemption process.
- `completeRedemption`: Completes the `esMOZ` redemption process.
- `convertToEsMOZ`: Converts `MOZ` tokens to `esMOZ`.
- `submitMultipleVerifications`: Submits multiple verification tasks.
- `claimMultipleRewards`: Claims multiple rewards for verification tasks.
- `convertToMOZ`: Converts `esMOZ` tokens and burns OG NFTs to redeem `MOZ`.

4 Findings

EMO-1 Incorrect Usage of `_pauseConvertToMOZ` in `convertToMOZ` Function

Severity: Medium

Status: Fixed

Code Location:

contracts/esMOZ.sol#304

Descriptions:

In `convertToMOZ` function, the variable `_pauseConvertToMOZ` is used in a manner inconsistent with its intended purpose.

The naming of `_pauseConvertToMOZ` implies that it is a flag indicating whether the "convert to MOZ" feature is paused.

However, the require statement:

```
function convertToMOZ(uint256[] memory _tokenIDs) public {
    require(_pauseConvertToMOZ, "Convert is currently inactive");
    //...//
}
```

allows execution when `_pauseConvertToMOZ == true`, which contradicts the expected behavior of a "pause"

Suggestion:

- Update the require condition to ensure that the function executes only when `_pauseConvertToMOZ` is false.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

EMO-2 Potential Array Out-of-Bounds Access

Severity: Minor

Status: Fixed

Code Location:

contracts/esMOZ.sol#200,220

Descriptions:

In the cancelRedemption and completeRedemption functions:

```
  require(_redemptionActive, "Redemption is currently inactive");
  RedemptionRequestExt storage request = _extRedemptionRequests[msg.sender]
[index];
```

This access could cause an array out-of-bounds error if index exceeds the array length.

Suggestion:

Add a check to ensure that the `index` is within bounds before accessing the array.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LBA-1 Single-step Ownership Transfer Can be Dangerous

Severity: Medium

Status: Fixed

Code Location:

contracts/LumozBaseAirdrop.sol#8

Descriptions:

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. If the admin permissions are given to the wrong address within this function, it will cause irreparable damage to the contract. Below is the official documentation explanation from OpenZeppelin https://docs.openzeppelin.com/contracts/4.x/api/access

`Ownable is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests but projects with production concerns are likely to outgrow it.`

The `LumozBaseAirdrop` contract inherits from the `OwnableUpgradeable` contract.

```
contract LumozBaseAirdrop is OwnableUpgradeable, PausableUpgradeable {
```

In this contract, transferring ownership is a single-step process, which poses the aforementioned risk. https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/OwnableUpgradeable.sol#L102-L118

Suggestion:

It is recommended to use the `Ownable2StepUpgradeable` contract.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LCC-1 Signature Replay Attack

Severity: Major

Status: Fixed

Code Location:

contracts/LicenseClaimContract.sol#67-91

Descriptions:

This claim() function enables users to claim NFT by providing required data and valid signatures. The issue here is that if a user has a valid signature, a malicious attacker can construct arrays with identical recordIndex , receiver , and claimCount values as function parameters and pass them to the claim() function. In this case, SignatureChecker.isValidSignatureNow() will validate the signature successfully each time. As a result, the attacker could ultimately mint multiple NFTs through NodeLicense(nodeLicenseAddress).batchMint() .

```
function claim(uint256[] memory recordIndexes, address[] memory receivers, uint256[] memory claimCounts, bytes[] memory signatures) public payable {
    require(!isStop, "Contract closed");
    uint256 _timestamp = block.timestamp;
    _timestamp = _timestamp - _timestamp % signatureInterval;

    require(recordIndexes.length == receivers.length, 'Array Mismatch');
    require(claimCounts.length == receivers.length, 'Array Mismatch');
    require(signatures.length == receivers.length, 'Array Mismatch');
    uint256 totalClaimedCount = 0;
    for (uint i = 0; i < recordIndexes.length; i++) {
      uint256 recordIndex = recordIndexes[i];
      address receiver = receivers[i];
      uint256 claimCount = claimCounts[i];
      bytes32 messageHash = getMessageHash(getMessage(msg.sender, recordIndex, receiver, claimCount, _timestamp));
      require(SignatureChecker.isValidSignatureNow(verifier, messageHash, signatures[i]), "invalid signature");
      totalClaimedCount += claimCount;
      require(totalClaimedCount <= maxClaimedCount, 'Total Claimed Counts Over maxClaimedCount');
```

```
            claimedRecords[msg.sender].push(ClaimedRecord(recordIndex, msg.sender, receiver, claimCount, block.timestamp));
            emit ClaimedInfo(recordIndex, msg.sender, receiver, claimCount, block.timestamp);
        }

        NodeLicense(nodeLicenseAddress).batchMint(receivers, claimCounts);
    }
```

Suggestion:

It is recommended to mark signatures that have already been used.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LOG-1 Missing disableInitializers Call in Proxy Upgradeable Contract Constructor

Severity: Medium

Status: Fixed

Code Location:

contracts/LumozOGNFT.sol#40

Descriptions:

The protocol does not call disableInitializers in the constructor of the logic contract during initialization. This oversight introduces a severe risk, allowing potential attackers to initialize the implementation contract itself.

```
constructor(uint256 _max) {
    maxSupplyPerLevel = _max;
}
```

Suggestion:

It is recommended to include a call to disableInitializers in the constructor of the logic contract as recommended by OpenZeppelin.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LOG-2 Logical Flaw in Blacklist Check for `LumozOGNFT` Contract

Severity: Medium

Status: Fixed

Code Location:

contracts/LumozOGNFT.sol#70,76,82

Descriptions:

In the `LumozOGNFT` contract, there is a logical flaw in the implementation of the blacklist validation. The current:

```
require(!isBlackListed[msg.sender] || !isBlackListed[from]);
```

Suggestion:

Update the blacklist check to ensure that both msg.sender and from are not blacklisted.

```
require(!isBlackListed[msg.sender] && !isBlackListed[from]);
```

This ensures that any transaction involving a blacklisted address is appropriately restricted, maintaining the integrity of the blacklist functionality.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LTA-1 Use `safeTransfer()` instead of `transfer()`

Severity: Major

Status: Fixed

Code Location:

contracts/LumozTokenAirdrop.sol#24

Descriptions:

In the `claim()` function, the protocol calls `ERC20Upgradeable(tokenAddress).transfer()` to transfer tokens to `msg.sender` and expects a `bool` return value.

```
bool bResult = ERC20Upgradeable(tokenAddress).transfer(msg.sender, amount);
        require(bResult, 'transfer failed.');
```

The issue is that some tokens, such as USDT, are not standard ERC-20 implementations and do not return a `bool` value. As a result, expecting a `bool` here can lead to a denial of service (DoS), preventing users from claiming their assets.

Suggestion:

It is recommended to use `safeTransfer()` instead of `transfer()`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LTA-2 Redundant Address Validation in claims Function

Severity: Informational

Status: Fixed

Code Location:

contracts/LumozTokenAirdrop.sol#36

Descriptions:

In the `claims` function of the `LumozTokenAirdrop` contract, the validation `onlyValidAddress(msg.sender)` is unnecessary.

The likelihood of `msg.sender` being an invalid addres 0x0) is negligible, and the function already includes the following validation:

```
require(msg.sender == reviewAuthority);
```

Suggestion:

Consider removing the `ValidAddress(msg.sender)` check to simplify the function logic.

The existing `require(msg.sender == reviewAuthority)` validation is adequate for ensuring the correctness and security of the caller.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

NFA-1 Missing Return Value Check

Severity: Minor

Status: Fixed

Code Location:

contracts/node_manager/NodeFactory.sol#193

Descriptions:

In the `stakeEsMOZ()` function, the protocol calls `esMOZ(payable(esMOZAddress)).transferFrom()` to transfer funds from `msg.sender`.

```
// node stake esMOZ
        ZKVerifierNodeProtocol(nodeProtocol).stakeEsMOZ(_nodeManager, _amount);

        esMOZ(payable(esMOZAddress)).transferFrom(msg.sender, address(this), _amount);
        NodeManager nodeManager = NodeManager(_nodeManager);
        nodeManager.stakeEsMOZ(msg.sender, _amount);
```

According to the EIP-20 standard, the return value of `transfer()` and `transferFrom()` methods should be checked.

https://eips.ethereum.org/EIPS/eip-20 However, the protocol currently does not perform this check.

Suggestion:

It is recommended to implement a check for the return value of `transfer()` and `transferFrom()` to ensure the transfer was successful.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

LOG1-1 Incorrect Order of Validation Conditions in `claim` Function

Severity: Informational

Status: Fixed

Code Location:

contracts/LumozOGAirdrop.sol#12,13

Descriptions:

In the `claim` function of the `LumozOGAirdrop` and `LumozTokenAirdrop` contracts, the following validation conditions:

```
    require(!isClaimed(rootIndex, index), 'cl:already claimed');
    require(rootIndex <= rootCounts, 'Invalid Root Index');
```

are implemented in an incorrect order. If `rootIndex > rootCounts` , may execute unnecessary logic.

Suggestion:

Swap the order of the require conditions to ensure that the rootIndex validation is executed first, as it is a prerequisite for the validity of subsequent checks. The revised order should be:

```
require(rootIndex <= rootCounts, 'Invalid Root Index');
require(!isClaimed(rootIndex, index), 'cl:already claimed');
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

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.