ZRO Claim Security Review

Pashov Audit Group

Conducted by: Peakbolt, ast3ros, btk

June 15th 2024 - June 17th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **LayerZero-Labs/ZROClaim** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About ZRO Claim

The ClaimRemote contract facilitates cross-chain token claims by sending messages to a remote ZROClaimHub contract and processes token claims based on Merkle proofs, ensuring the donor's contribution before validating the claim. It includes rate limiting, fee quoting, and configurable claim parameters, requiring owner permission to set rate limits and verify message integrity.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 36ea2473e9b67aa6681dc1ebb368c7db6dcd0cd1

fixes review commit hash - f4871d4f330ddb6b3eacc89edbb4b3d25a7167e0

Scope

The following smart contracts were in scope of the audit:

- `ZROClaimCore`
- `ZROClaimHub`
- `ZROClaimSpoke`
- `IZROCLaim`

7. Executive Summary

Over the course of the security review, Peakbolt, ast3ros, btk engaged with LayerZero to review ZRO Claim. In this period of time a total of **10** issues were uncovered.

Protocol Summary

Protocol Name	ZRO Claim
Repository	https://github.com/LayerZero-Labs/ZROClaim
Date	June 15th 2024 - June 17th 2024
Protocol Type	Token claim

Findings Count

Severity	**Amount**
Critical	2
High	2
Medium	3
Low	3
Total Findings	**10**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Numerators are off by one	Critical	Resolved
[C-02]	Remote donations are stuck	Critical	Resolved
[H-01]	WithdrawDonation can incur max slippage	High	Resolved
[H-02]	_lzReceive reverts when there is a fee refund	High	Resolved
[M-01]	donateAndClaim() will not work for remote donate and claim	Medium	Resolved
[M-02]	DonateRemote will fail to deploy when any of the stargate tokens is not set	Medium	Resolved
[M-03]	Protocol assumes 1 Stable = 1 USD	Medium	Acknowledged
[L-01]	Precision loss in requiredDonation()	Low	Acknowledged
[L-02]	Donations in the native token can be not counted	Low	Resolved
[L-03]	quoteClaimCallback has 0 slippage tolerance	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Numerators are off by one

Severity

Impact: High

Likelihood: High

Description

`ClaimCore` constructor will set the numerators (`numeratorUsdc`, `numeratorUsdt`, `numeratorNative`) for computing the required donation amount to claim the ZRO token.

However, the numerators are off by a factor of 10 as the base10 exponent is incorrectly subtracted by 1 during initialization. This will reduce the donations required by a factor of 10, allowing anyone to claim 10x more ZRO than expected.

```
constructor(
        bytes32 _merkleRoot,
        address _donateContract,
        address _stargateUsdc,
        address _stargateUsdt,
        address _stargateNative,
        uint256 _nativePrice,
        address _owner
    ) Ownable(_owner) {
        merkleRoot = _merkleRoot;
        donateContract = IDonate(_donateContract);

        // TODO needs tests
        if (_stargateUsdc != address(0)) {
            //@audit this is off by a factor of 10
>>>         numeratorUsdc = 1 * 10 ** (IERC20Metadata(IOFT(_stargateUsdc).token
  ()).decimals() - 1);
        }

        if (_stargateUsdt != address(0)) {
            //@audit this is off by a factor of 10
>>>         numeratorUsdt = 1 * 10 ** (IERC20Metadata(IOFT(_stargateUsdt).token
  ()).decimals() - 1);
        }

        // native is always denominated in 18 decimals
        if (_stargateNative != address(0) && _nativePrice > 0) {
            //@audit this is off by a factor of 10
>>>         numeratorNative = _nativePrice * 10 ** (18 - 1);
            // Validate this is an actual native pool, eg. NOT WETH
            if (IOFT(_stargateNative).token() != address(0)) {
                revert InvalidNativeStargate();
            }
        }
    }
```

Recommendations

Remove the subtraction by 1 in the initialization of numerators.

[C-02] Remote donations are stuck

Severity

Impact: High

Likelihood: High

Description

Users interact with `RemoteDonate` to perform donations on non-Ethereum chains. And the donation will be withdrawn remotely to the `donationReceiver` on Ethereum chain via Stargate cross-chain transfer.

Due to the cross-chain transfer, an LZ fee is required to be paid via `msg.value` while transferring the `donationAmount` to Ethereum chain.

The issue is that `donationAmount` is assigned the value of `address(this).balance` , which actually includes the `msg.value` meant for the transfer across chain.

That will cause the `IOFT(stargate).send{ value: msg.value + donationAmountNative }` to fail as it is sending more than `address(this).balance`.

The impact of this is that withdrawals for remote donations will always fail, causing them to be stuck within the contract.

```
function withdrawDonation
      (Currency _currency, uint256 _minAmount) external payable {
        address stargate;
        uint256 donationAmount;
        uint256 donationAmountNative;

        if (_currency == Currency.USDC && stargateUsdc != address(0)) {
            stargate = stargateUsdc;
            donationAmount = tokenUsdc.balanceOf(address(this));
        } else if (_currency == Currency.USDT && stargateUsdt != address(0)) {
            stargate = stargateUsdt;
            donationAmount = tokenUsdt.balanceOf(address(this));
        } else if (_currency == Currency.Native && stargateNative != address
          (0)) {
            stargate = stargateNative;
        //@audit this would have included the LZ fee (msg.value) as well
>>>         donationAmount = address(this).balance;
            donationAmountNative = donationAmount;
        } else {
            revert UnsupportedCurrency(_currency);
        }//ok

        // sends via taxi
        bytes memory emptyBytes = new bytes(0);
        SendParam memory sendParams = SendParam({
            dstEid: remoteEid, // only send to ethereum
            to: bytes32(uint256(uint160(donationReceiver))),
            amountLD: donationAmount,
            minAmountLD: _minAmount,
            extraOptions: emptyBytes,
            composeMsg: emptyBytes,
            oftCmd: emptyBytes // type taxi
        });//ok

        // combine the msg value in addition to the donation amount
        // in non native currencies this will just be 0
        // solhint-disable-next-line check-send-result

       //@audit this will always fail since it is sending more than address
       //(this).balance
 >>>    IOFT(stargate).send{ value: msg.value + donationAmountNative }(
            sendParams,
            MessagingFee(msg.value, 0),
            msg.sender // refund any excess native to the sender
        );

        emit DonationWithdrawn(_currency, donationReceiver, donationAmount);
    }
```

Recommendations

Make the following change to calculate the actual donation amount for transfer.

```
} else if (_currency == Currency.Native && stargateNative != address
          (0)) {
            stargate = stargateNative;
-             donationAmount = address(this).balance;
+             donationAmount = address(this).balance - msg.value;
            donationAmountNative = donationAmount;
        } else {
```

8.2. High Findings

[H-01] WithdrawDonation can incur max slippage

Severity

Impact: High

Likelihood: Medium

Description

The `withdrawDonation` function in `DonateRemote` is a permissionless function that allows anyone to transfer donations to the designated donation receiver. The slippage protection parameter, `minAmount`, is set by the sender. This means that any account can set the `_minAmount` to 0, which allows the assets sent to the local chain to incur maximum slippage loss.

```
function withdrawDonation
      (Currency _currency, uint256 _minAmount) external payable {
        ...
        // sends via taxi
        bytes memory emptyBytes = new bytes(0);
        SendParam memory sendParams = SendParam({
            dstEid: remoteEid, // only send to ethereum
            to: bytes32(uint256(uint160(donationReceiver))),
            amountLD: donationAmount,
            minAmountLD: _minAmount, // @audit can be set to 0
            extraOptions: emptyBytes,
            composeMsg: emptyBytes,
            oftCmd: emptyBytes // type taxi
        });
        ...
    }
```

Recommendations

`withdrawDonation` in DonateRemote should be a permissioned function to allow setting the `_minAmount` parameter to a safe value.

[H-02] `_lzReceive` reverts when there is a fee refund

Severity

Impact: High

Likelihood: Medium

Description

When sending a message to the remote `ZROToken` for transferring the balance of ZROToken to users, if the fee exceeds the required amount, the surplus is refunded to `address(this)` (the `ClaimLocal` contract). However, the `ClaimLocal` contract does not have a `fallback` or `receive` function, preventing it from receiving ETH. This causes the `_lzReceive` function to always revert, preventing users from claiming tokens on remote chains.

```
function _lzReceive(
        Origin calldata _origin,
        bytes32 /*_guid*/,
        bytes calldata _payload,
        address /*_executor*/,
        bytes calldata /*_extraData*/
    ) internal override {
        (address user, uint256 zroAmount, address to) = abi.decode(_payload,
          (address, uint256, address));
        ...

        // solhint-disable-next-line check-send-result
        IOFT(zroToken).send{ value: msg.value }(sendParams, MessagingFee
        //(msg.value, 0), address(this)); // @audit refund fee cannot be received by C

        emit ClaimRemote(user, availableToClaim, _origin.srcEid, to);
    }
```

The LayerZero Endpoint contract uses SendLibrary, which uses `TransferLibrary` for low-level calls to return the fees.

```
library Transfer {
    ...

    function native(address _to, uint256 _value) internal {
        if (_to == ADDRESS_ZERO) revert Transfer_ToAddressIsZero();
        (bool success, ) = _to.call{ value: _value }("");
        if (!success) revert Transfer_NativeFailed(_to, _value);
    }

    ...
}
```

link

Recommendations

Add a `receive` function to the ClaimLocal contract to enable it to accept ETH refunds.

```
+    receive() external payable {}
```

8.3. Medium Findings

[M-01] `donateAndClaim()` will not work for remote donate and claim

Severity

Impact: Medium

Likelihood: Medium

Description

Users can utilize `donateAndClaim()` to make a donation in native token via `msg.value` and then perform a ZRO claim.

However, this will not work as it does not pass the LZ fee in the `value` parameter for `claim()` to perform the remote claim.

```
function donateAndClaim(
        Currency currency,
        uint256 amountToDonate,
        uint256 _zroAmount,
        bytes32[] calldata _proof,
        address _to,
        bytes calldata _extraBytes
    ) external payable returns (MessagingReceipt memory receipt) {
        // move tokens into the wrapper to forward to the donate contract
        if (currency == Currency.USDC && stargateUsdc != address(0)) {
            tokenUsdc.safeTransferFrom(msg.sender, address
              (this), amountToDonate);
        } else if (currency == Currency.USDT && stargateUsdt != address(0)) {
            tokenUsdt.safeTransferFrom(msg.sender, address
              (this), amountToDonate);
        } else if (currency == Currency.Native && stargateNative != address
          (0)) {
            // native do nothing
        } else {
            // sanity just in case somehow a different currency is somehow
            // passed
            revert IDonate.UnsupportedCurrency(currency);
        }

        // donate
        IDonate(donateContract).donate{ value: msg.value }
          (currency, amountToDonate, msg.sender);

        // claim

        //@audit need to pass in the gas value for remote claim
>>>     return IClaim(claimContract).claim
  (msg.sender, currency, _zroAmount, _proof, _to, _extraBytes);
    }
```

```
function _claim(
        address _user,
        Currency _currency,
        uint256 _zroAmount,
        bytes32[] calldata _proof,
        address _to,

    ) internal returns (MessagingReceipt memory receipt) {
        // step 1: assert claimer has donated sufficient amount
        assertDonation(_currency, _user, _zroAmount);

        // step 2: validate the proof is correct
        if (!_verifyProof(_user, _zroAmount, _proof)) revert InvalidProof();

        // step 3: generate the message and options for the crosschain call
        bytes memory payload = abi.encode(_user, _zroAmount, _to);
        bytes memory options = combineOptions
          (remoteEid, MSG_TYPE_CLAIM, _extraOptions);

        // step 4: send the message cross chain
        // solhint-disable-next-line check-send-result

        //@audit this will use the donated native tokens for cross-chain
        // messaging as well
>>>     receipt = endpoint.send{ value: msg.value }(
            MessagingParams(remoteEid, _getPeerOrRevert
              (remoteEid), payload, options, false),
            payable(_user)
        );

        emit ZRORequested(_user, _zroAmount, _to);
    }
```

Recommendations

For remote donations and claims in native tokens, the users should also provide the LZ sending fee in addition to the donated native currency. In that scenario, `DonateAndClaim()` will pass `amountToDonate()` to the donate contract, while the LZ fee will be passed to the claim contract.

[M-02] `DonateRemote` will fail to deploy when any of the stargate tokens is not set

Severity

Impact: Medium **Likelihood:** Medium

Description

Both `DonateCore` and `DonateLocal` are implemented to allow the donation contracts to work even when any of the `stargateUsdc`, `stargateUsdt`, and `stargateNative` addresses are not set. That means these contracts have the flexibility to support any combination of the Stargate tokens and do not require all to be enabled.

However, `DonateRemote` constructor is implemented to require all the Stargate tokens to be enabled. Without any of them, the constructor will revert and fail to initialize. This is incorrect if the intention is to maintain flexibility like `DonateLocal`.

Also, the `IOFT(_stargateNative).token() != address(0))` is redundant as it has been performed in `DonateCore`.

```
contract DonateRemote is DonateCore {
    constructor(
        uint32 _remoteEid,
        address _donationReceiver,
        address _stargateUsdc,
        address _stargateUsdt,
        address _stargateNative
    ) DonateCore
      (_donationReceiver, _stargateUsdc, _stargateUsdt, _stargateNative) {
        remoteEid = _remoteEid;

        //@audit missing != address(0) checks for the following
>>>     IERC20(IOFT(_stargateUsdc).token()).forceApprove(_stargateUsdc, type
  (uint256).max);
>>>     IERC20(IOFT(_stargateUsdt).token()).forceApprove(_stargateUsdt, type
  (uint256).max);

        //@audit this is redundant as this is checked in DonateCore
>>>     if (IOFT(_stargateNative).token() != address(0)) {
            revert InvalidNativeStargate();
        }
    }
```

Recommendations

Within the `DonateRemote` constructor,

- Add the conditional `!= address(0)` checks for the respective call to the stargate tokens.
- Remove the redundant check for `_stargateNative`.

[M-03] Protocol assumes 1 Stable = 1 USD

Severity

Impact: Medium

Likelihood: Medium

Description

In the ClaimCore contract constructor, the prices for USDC and USDT are hardcoded to 1 USD:

```
if (_stargateUsdc != address(0)) {
            numeratorUsdc = 1 * 10 ** (IERC20Metadata(IOFT(_stargateUsdc).token
              ()).decimals() - 1);
        }

        if (_stargateUsdt != address(0)) {
            numeratorUsdt = 1 * 10 ** (IERC20Metadata(IOFT(_stargateUsdt).token
              ()).decimals() - 1);
        }
```

This assumption can lead to problems during stablecoin volatility events. For instance, if users are donating 1000 USDC (equivalent to 1000 USD) in exchange for 1000 ZRO, a depeg event where USDC drops to 0.8 USD means users can contribute 1000 USDC (now worth 800 USD) for the same 1000 ZRO.

Recommendations

Consider using a chainlink oracle to fetch the stables prices.

8.4. Low Findings

[L-01] Precision loss in `requiredDonation()`

`requiredDonation()` determines the number of required donations in USDC/USDT precision (6 decimals), which is lower than ZRO's precision (18 decimals). This will result in a precision loss, which could result in less USDC/USDT required for donation.

This can be fixed by rounding up the required donation amount.

```
function requiredDonation(uint256 _zroAmount) public view returns
      (Donation memory) {
        uint256 usdc = (_zroAmount * numeratorUsdc) / DENOMINATOR;
        uint256 usdt = (_zroAmount * numeratorUsdt) / DENOMINATOR;
        uint256 native = (_zroAmount * numeratorNative) / DENOMINATOR;
        return Donation(usdc, usdt, native);
    }
```

[L-02] Donations in the native token can be not counted

The `donate` function allows users to donate and transfer tokens to the `DonateCore` contract. When `_currency` is specified to tokens other than the native token, it transfers the token amount to the `Donate` contract. However, in such cases, it does not check if `msg.value == 0`. If the donor also sends native tokens in the donation transaction, the donation in native tokens is not accounted for because the branch `else if (_currency == Currency.Native && stargateNative != address(0))` is never executed. This results in the native tokens being ignored.

It is recommended to check if `msg.value > 0` and increase the donations in native tokens accordingly.

```
function donate
    //(Currency _currency, uint256 _amount, address _beneficiary) external payable { /
        if (_currency == Currency.USDC && stargateUsdc != address(0)) {
            tokenUsdc.safeTransferFrom(msg.sender, address(this), _amount);
            donations[stargateUsdc][_beneficiary] += _amount;
        } else if (_currency == Currency.USDT && stargateUsdt != address(0)) {
            tokenUsdt.safeTransferFrom(msg.sender, address(this), _amount);
            donations[stargateUsdt][_beneficiary] += _amount;
        } else if (_currency == Currency.Native && stargateNative != address
          (0)) {
            if (msg.value != _amount) revert InsufficientMsgValue();
            donations[stargateNative][_beneficiary] += _amount;
        } else {
            // sanity just in case somehow a different currency is somehow
            // passed
            revert UnsupportedCurrency(_currency);
        }
        ...
    }
```

[L-03] `quoteClaimCallback` has 0 slippage tolerance

The `quoteClaimCallback()` function currently quotes the native amount required to send ZRO across chains as follows:

```
function quoteClaimCallback(
      uint32_dstEid,
      uint256_zroAmount
    ) external view returns (MessagingFee memory msgFee
        bytes memory emptyBytes = new bytes(0);
        SendParam memory sendParams = SendParam({
            dstEid: _dstEid,
            to: addressToBytes32(msg.sender),
            amountLD: _zroAmount,
            minAmountLD: _zroAmount,
            extraOptions: emptyBytes,
            composeMsg: emptyBytes,
            oftCmd: emptyBytes
        });
        return IOFT(zroToken).quoteSend(sendParams, false);
    }
```

In this implementation, the `_zroAmount` is passed directly as `minAmountLD`. This implies that the callback will revert for any slippage, even as small as 0.001. Consider adding a `minZroAmount` parameter to the callback and passing it as `minAmountLD` to allow for some tolerance. Please note that the same issue exists in `ClaimLocal._lzReceive()`.