PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero ZRO Claim

18 Jun 2024

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team. Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for LayerZero ZRO Claim on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	LayerZero ZRO Claim
URL	https://layerzero.network/
Platform	Ethereum
Language	Solidity
Preliminary	https://github.com/LayerZero-Labs/ZROClaim/tree/dde1ee43fa81ff22daa0f8e2f02ffede7d2a784d/contracts
Resolutions	https://github.com/LayerZero-Labs/ZROClaim/tree/bf965039f2793cfcc91d265a87c6288302ed4b96/contracts

1.2 Contracts Assessed

Name	Contract	Live Code Match
DonateAndClaim		PENDING
ClaimCore		PENDING
ClaimLocal		PENDING
ClaimRemote		PENDING
DonateCore		PENDING
DonateLocal		PENDING
DonateRemote		PENDING

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● Governance	-	-	-	-
● High	1	1	-	-
● Medium	-	-	-	-
● Low	3	3	-	-
● Informational	6	5	1	-
Total	**10**	**9**	**1**	**-**

Classification of Issues

Severity	Description
● **Governance**	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
● **High**	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● **Medium**	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● **Low**	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● **Informational**	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 Global

ID	Severity	Summary	Status
01	INFO	Various functions can be called with a msg.value even if they do not need one, possibly confusing people to send gas tokens with these functions	✓ RESOLVED

1.3.2 DonateAndClaim

ID	Severity	Summary	Status
02	INFO	Typographical issues and gas optimizations	✓ RESOLVED

1.3.3 ClaimCore

ID	Severity	Summary	Status
03	HIGH	Native value logic is mispriced requiring at least 0.1 ETH to be donated per ZRO token claimed, significantly more than the desired $0.1 in ETH	✓ RESOLVED
04	LOW	Native prices cannot be updated once configured	✓ RESOLVED
05	INFO	Typographical issues	✓ RESOLVED

1.3.4 ClaimLocal

ID	Severity	Summary	Status
06	LOW	A configured but malicious DVN or executor could take all ETH in this contract by increasing their fee	✓ RESOLVED
07	INFO	Typographical issues and gas optimizations	✓ RESOLVED

1.3.5 ClaimRemote

ID	Severity	Summary	Status
08	INFO	Typographical issues	PARTIAL

1.3.6 DonateCore

No issues found.

1.3.7 DonateLocal

No issues found.

1.3.8 DonateRemote

ID	Severity	Summary	Status
09	LOW	withdrawDonation stargate slippage protection can be freely chosen by an untrusted caller	✓ RESOLVED
10	INFO	Typographical issues and gas optimizations	✓ RESOLVED

2 Findings

2.1 Global

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.2 Issues & Recommendations

Issue #01	Various functions can be called with a msg.value even if they do not need one, possibly confusing people to send gas tokens with these functions
Severity	● INFORMATIONAL
Description	Many of the functions within the LayerZero claim architecture allow for the user to submit a `msg.value`. This is done to either pay for a donation in the native gas token or the LayerZero message fee.

1. `donate` can be called with a `msg.value` even if the donated currency is not `NATIVE`.

DonateCore::L70-71

```
} else if (_currency == Currency.Native && stargateNative
!= address(0)) {
  if (msg.value != _amount) revert InsufficientMsgValue();
```

The `donate` function allows for users to donate `USDT`, `USDC` or native gas tokens for a specific wallet. This increments the recipient their tracked donation value, making them eventually eligible to claim their `ZRO` tokens. This `donate` function correctly validates that sufficient native gas tokens are provided whenever the `Currency.Native` is provided as the `Currency` parameter, as visible in the snippet above. However, when a different currency is provided, such validation is not present. This means that in those other cases, the caller is still able to provide a `msg.value` accidentally. This `msg.value` would then remain unrecorded and not appear as a donation.

2. `withdrawDonation` is payable allowing people to provide a `msg.value`, even though this value won't be explicitly donated for a specific recipient

DonateLocal::L25

```
function withdrawDonation(Currency _currency, uint256
/*_minAmount*/) external payable
```

Even though the `withdrawDonation` function requires a `msg.value` to be passed for the `DonateRemote` contract, this is not necessary within the `DonateLocal` contract. Within `DonateRemote`, it is used to pay the gas cost for the LayerZero message. Since no LayerZero message occurs within `DonateLocal`, there's absolutely no reason for people to pass a gas value within it's `withdrawDonation` function. If this were to be passed, it would still be sent to the `donationReceiver` but the downside is that this donation would never be allocated to a specific recipient.

3. `claim` within the `ClaimLocal` contract is payable allowing people to provide a `msg.value`, even though this value won't be used and will need to be withdrawn by the team.

ClaimLocal::L73 and ClaimLocal::L85

```
) external payable returns (MessagingReceipt memory
receipt) {
  ) external payable onlyDonateAndClaim returns (Messag-
ingReceipt memory receipt) {
```

The `claim` functions within `ClaimLocal` also are marked with `payable` similar to the remote functions where this is necessary to cover the LayerZero message fee. However, here this gas will be stuck within the contract until the contract `owner` withdraws it. It could be accidentally sent into here if users provide too much value to `donateAndClaim` or simply accidentally provide value here.

Recommendation

1. Consider validating that `msg.value` is zero within the `USDT` and `USDC` `Currency` types:

```
if(msg.value != 0) revert UnexpectedMsgValue();
```

2. Consider validating that `msg.value` is zero.

3. Consider explicitly validating that `msg.value` is zero within `_claim` in `ClaimLocal`. `DonateAndClaim` needs to be updated as well to ensure a zero value can be sent on this call path.

Resolution ✓ RESOLVED

2.2 DonateAndClaim

The `donateAndClaim` contract allows for users who are eligible for the `ZRO` airdrop to claim it by calling the `donateAndClaim` function. To be able to call it, they must be included in the claim contract's merkle tree. They must furthermore "donate" a number of either `USDC`, `USDT` or gas tokens (eg. `ETH`) for every `ZRO` token they wish to claim. These donations can be withdrawn to the configured donations receiver, on Ethereum. For claims on different chains than ethereum, the donation will be sent to the donation receiver via Stargate.

2.2.1 Privileged Functions

None.

2.2.2 Issues & Recommendations

Issue #02	Typographical issues and gas optimizations
Severity	● INFORMATIONAL
Description	Lines 79 and 84 `donateNativeAmount = 0;` This value is still zero at this stage. This operation therefore simply wastes some gas.
Recommendation	Consider fixing the typographical issues and gas optimizations.
Resolution	✓ RESOLVED

2.3 ClaimCore

The `ClaimCore` contract is extended by either `ClaimLocal` or `ClaimRemote` and contains the business logic which validates that a user is actually included in the airdrop merkle tree. It therefore ensures that a claiming user is actually eligible for an airdrop. It furthermore queries the donation contract to ensure that the user has donated sufficient tokens to claim their airdrop.

Throughout this audit we assume that the `ZRO` token is `18` decimals. This is very reasonable and expected to be the case. If not, various portions of the codebase will malfunction.

It should be noted that the `owner` of this contract has significant control over the claim and donate process, as the `owner` can reconfigure eg. the merkle root to a malicious version, alongside with simply withdrawing the `ZRO` tokens within `ClaimLocal` using the `withdrawZro` function. Investors in the `ZRO` token should therefore be mindful that this `owner` role must be sufficiently safeguarded, in eg. a carefully chosen multisig.

Reconfiguring the merkle root could be a desired feature as it would allow for the airdrops to be increased over time. Any incremental value can them be claimed again, as the contract properly allows for this.

It should finally be noted that this audit was conducted after other audit(s), meaning that the code which was handed to Paladin already had several issues fixed. The resolved issues from those audit(s) have not been repeated within this report, as to keep the report brief and focused on the actual code revision which was handed to us, compared to the original revision which had a few more vulnerabilities which are now already fixed.

2.3.1 Privileged Functions

- `setDonateAndClaim`
- `setMerkleRoot`
- `transferOwnership`
- `renounceOwnership`

2.3.2 Issues & Recommendations

Issue #03	Native value logic is mispriced requiring at least 0.1 ETH to be donated per ZRO token claimed, significantly more than the desired $0.1 in ETH
Severity	● HIGH SEVERITY

Description	Line 56
	
	`numeratorNative = _nativePrice * 10 ** (18 - 1);`
	
	Line 78
	
	`uint256 native = ((_zroAmount * numeratorNative) + DENOMINATOR - 1) / DENOMINATOR;`
	
	During contract deployment, the contract allows for the provisioning of a native price. This price is used to adjust the required amount of ETH and other native donations so that approximately $0.1 is required to be donated per ZRO token claimed.
	
	However, due to a mathematical error within the code snippets above, the lowest that this donation proportion can be configured at is actually `0.1` ETH per ZRO token. At current prices this would be about $350 instead of the desired $0.1.

Recommendation	Consider adjusting the `numeratorNative` calculation. Line 56 `numeratorNative = 10 ** (18 - 1) / _nativePrice;`
Resolution	✓ RESOLVED The recommendation has been introduced.

Issue #04	Native prices cannot be updated once configured
Severity	● LOW SEVERITY
Description	Even though the `nativePrice` can be initially configured at eg. $3,500 ETH/USD, this `nativePrice` cannot be adjusted afterwards. This means that if ETH or another native currency reduces significantly in value over the claiming period, eg. to $1,500, claiming cost will be significantly cheaper in the native currency than in the USDC or USDT currency. This is rated as low as we believe the LayerZero team may be comfortable reducing their $/ZRO token donation requirement in such an event. They may be fine with receiving less ETH in dollar terms if its value goes down.
Recommendation	If desired, consider allowing for the `nativePrice` to be reconfigurable.
Resolution	✓ RESOLVED The client has indicated they are comfortable with taking on this price risk. This is as designed to keep the contract simple. No changes were made.

Issue #05	Typographical issues
Severity	INFORMATIONAL

Description

Lines 55-61

```
if (_stargateNative != address(0) && _nativePrice > 0) {
  numeratorNative = _nativePrice * 10 ** (18 - 1); // -1
for 10%
  // Validate this is an actual native pool, eg. NOT WETH
  if (IOFT(_stargateNative).token() != address(0)) {
    revert InvalidNativeStargate();
  }
}
```

The `.token()` check does not occur in the above snippet if `nativePrice` is configured to be zero. However, a `stargateNative` contract was still provided in this instance. It is unclear why this combination of settings should be permitted.

Consider adjusting the above snippet as follows:

```
if (_stargateNative != address(0)) {
  if (_nativePrice == 0) revert InvalidNativePrice(_nat-
ivePrice);
  numeratorNative = _nativePrice * 10 ** (18 - 1); // -1
for 10%
  // Validate this is an actual native pool, eg. NOT WETH
  if (IOFT(_stargateNative).token() != address(0)) {
    revert InvalidNativeStargate();
  }
}
```

Line 73

```
* @return donation QuoteDonation amount in native and
stable coin.
```

The word *QuoteDonation* is odd to us. Perhaps the writer accidentally combined two words here?

Finally, the initial code we audited rounded the donation amount up, which is desired. However, later revisions of the code started rounding it down again. Though there is no direct downside to this and we would likely also round down because it simplifies the code, we do want to point out that rounding against the favor of the user, up in this case, is a good design pattern to avoid exploits.

Recommendation	Consider fixing the typographical issues.
Resolution	✓ RESOLVED

2.4 ClaimLocal

The `ClaimLocal` contract is the version of `ClaimCore` which is deployed on the Ethereum network. It allows for direct claiming of `ZRO` tokens by users who are included in the merkle tree.

It furthermore defines the business logic for sending requested `ZRO` tokens to networks with a `ClaimRemote` deployment.

2.4.1 Privileged Functions

- `setDonateAndClaim`
- `setMerkleRoot`
- `withdrawZro`
- `withdrawNative`
- `setPeer`
- `setDelegate`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #06	**A configured but malicious DVN or executor could take all ETH in this contract by increasing their fee**
Severity	● LOW SEVERITY
Description	Line 141 `IOFT(zroToken).send{ value: address(this).balance }(` The `ClaimLocal` contract blindly subsidizes the LayerZero gas cost to bridge back the `zro` token from Ethereum into the various remote chains. The actual ETH that is consumed here depends on the pricing parameters of the configured DVNs and `executor` within the `zroToken`, if any one of these independent parties decides to increase their fee significantly, eg. to the full contract balance, there's no validation that prevents the full balance from being drained here.
Recommendation	Consider being careful with the DVN and `executor` configuration, though we have no doubt in this. Consider whether capping this value might make sense. Alternatively, simply keeping the ETH within this contract to a reasonable amount suffices as well. That way if it happens, the impact would be minimal.
Resolution	✓ RESOLVED The client has indicated they are dilligent with their configurations so that this will not happen. They've furthermore indicated that they will keep a reasonable and not excessive balance within this contract. No changes were made.

Issue #07	Typographical issues and gas optimizations
Severity	● INFORMATIONAL

Description	Lines 50 and 55
	
	`function withdrawZro(address _to, uint256 _amount) external onlyOwner`
	`function withdrawNative(address _to, uint256 _amount) external onlyOwner`
	
	Both of these functions lack an event.
	
	Line 54
	
	`// Withdraw accumulated b->a msg values that are refunded into here`
	
	This is not the only value that can be withdrawn via this function: Additional gas tokens sent into this contract to subsidize the message callback can also be withdrawn here.
	
	Line 77
	
	`// Allows the wrapper to apss in a user address to claim on their behalf`
	
	This should spell *pass*.
	
	Line 151
	
	`// prevents a user from claiming more than once`
	
	Strictly speaking multiple claims are permitted, however, claiming the same allocation twice is not permitted.
	
	Lines 152-153
	
	`if (_zroAmount <= zroClaimed[_user]) revert AlreadyClaimed(_user);`
	`uint256 alreadyClaimed = zroClaimed[_user];`
	
	These two lines could be inverted to avoid fetching from storage twice.

Recommendation	Consider fixing the typographical issues and gas optimizations.
Resolution	✓ RESOLVED

2.5 ClaimRemote

The `ClaimRemote` contract is quite similar to the `ClaimLocal` contract. However, instead of directly sending the claimed `ZRO` tokens to the user, it sends a LayerZero message to the `ClaimLocal` contract on Ethereum requesting `ZRO` tokens for the user. The `ClaimLocal` contract will then call the `ZRO` contract, which is an `OFT`, and call the `OFT send` function to send the `ZRO` tokens to the remote chain and into the claimer's wallet.

Note that the validation to avoid double-claiming occurs within the `ClaimLocal` contract, meaning `claim` can be called multiple times at the remote. Users should only call it once however, since only the first request will be fulfilled. The idea is that the same merkle root is shared amongst all chains allowing for a user to claim once anywhere, but not once on two different chains for example.

A rate limit on the messages sent to mainnet was introduced during the resolutions, configurable by the owner via `setRateLimits`. The implementation of this rate limiter is out of scope for our audit.

2.5.1 Privileged Functions

- `setDonateAndClaim`
- `setMerkleRoot`
- `setEnforcedOptions`
- `setPeer`
- `setDelegate`
- `setRateLimits`
- `transferOwnership`
- `renounceOwnership`

2.5.2 Issues & Recommendations

Issue #08	Typographical issues
Severity	● INFORMATIONAL

Description

Lines 5 and 20

```
import { SafeERC20, IERC20 } from "@openzeppelin/contra-
cts/token/ERC20/utils/SafeERC20.sol";
using SafeERC20 for IERC20;
```

This import appears unused.

Line 12

```
import { OAppOptionsType3 } from "@layerzerolabs/lz-evm-
-oapp-v2/contracts/oapp/libs/OAppOptionsType3.sol";
```

This has already been imported.

Finally, this code often mentions a `ZROClaimHub`, a contract which we do not see within this scope. Those comments may be outdated.

Lines 111-114

```
receipt = endpoint.send{ value: msg.value }(
  MessagingParams(remoteEid, _getPeerOrRevert(remoteEid-
), payload, options, false),
  payable(_user)
);
```

It may be considered slightly more idiomatic to re-use the already existing `_lzSend` internal function for this, though we admit the current implementation will save a bit of gas.

Line 121

```
(address user, uint256 zroAmount, address to) = abi.-
decode(_payload, (address, uint256, address));
```

It would have been cleaner to encode all `address` values to `bytes32`, then this contract would be compatible with non-evm chains as well. However, we don't believe it's worth changing that at this stage.

Finally, it may make sense to prevent `address(0)` from being set as the `to` address, as the message will succeed but it is likely that the eventual `ZRO` token on this network will revert a transfer to this address. Donations to this

	address could then also be disabled for consistency, though that is not strictly required.
Recommendation	Consider fixing the typographical issues.
Resolution	● PARTIALLY RESOLVED Some of the above typograhical issues have been resolved.

2.6 DonateCore

The `DonateCore` contract is extended by either `ClaimLocal` or `ClaimRemote` and contains the business logic which allows for users to donate `USDC`, `USDT` or the chain's local gas token to the configured donation receiver.

A donation amount proportional to the amount of `ZRO` tokens a user is eligible for is required for said user to be able to claim these `ZRO` tokens.

For remote chains, certain of the donation token kinds might be disabled, eg. you may only be able to donate the local gas token.

The actual required proportions of each of the three coins is configured within the `ClaimCore` within the `numeratorUsdc`, `numeratorUsdt` and `numeratorNative` variables.

Next, the donation contracts allow for people to donate to other wallets. This means that someone else could donate to your wallet to allow you to claim without further donations.

It should be noted that the three donation currencies are not cummulative. This means that even if people donate to your wallet in all three currencies, you can only use the donations for one of those three currencies to claim your tokens.

All donation amounts are stored within the storage of the donation contracts, which allows for the `ClaimLocal` and `ClaimRemote` contracts to access this storage and check whether enough tokens have been donated to your wallet.

2.6.1 Privileged Functions

None.

2.6.2 Issues & Recommendations

No issues found.

2.7 DonateLocal

The `DonateLocal` contract extends the `DonateCore` contract and is supposed to be deployed on the Ethereum mainnet. Alongside the `DonateCore` logic mentioned above, it allows for anyone to call `withdrawDonation`, which transfers the `USDC`, `USDT` or `ETH` balance to the configured `donationReceiver`.

It should be noted that if the `donationReceiver` is a contract, it should be able to receive `ETH` native tokens.

2.7.1 Privileged Functions

None.

2.7.2 Issues & Recommendations

No issues found.

2.8 DonateRemote

The `DonateRemote` contract extends the `DonateCore` contract and is supposed to be deployed on all networks other than Ethereum. Alongside the `DonateCore` logic mentioned above, it allows for anyone to call `withdrawDonation`, which uses LayerZero to send the `USDC`, `USDT` and native gas token via stargate to the configured `donationReceiver` on the Ethereum mainnet.

Both the remote and local version of the donation contract will therefore eventually send the donations to the same receiver on mainnet, either directly or indirectly.

2.8.1 Privileged Functions

None.

2.8.2 Issues & Recommendations

Issue #09	**withdrawDonation stargate slippage protection can be freely chosen by an untrusted caller**
Severity	● LOW SEVERITY
Description	Line 41 `function withdrawDonation(Currency _currency, uint256 _minAmount) external payable {` As stargate levies a fee, the `DonateRemote` function introduces a slippage check that validates that this fee is not excessive: The `_minAmount` can be provided by the caller of `withdrawDonation` to validate that at least that amount is actually sent to the `donationReceiver` after fees. However, the issue here is that anyone can call `withdrawDonation`, and an untrusted caller may very well set this value to zero, fully mitigating the benefit of this check. If this is done, such donations transfers may pass with fees larger as what the LayerZero team would have liked to permit.
Recommendation	Consider making this function privileged, or allowing the admin to configure this slippage amount.
Resolution	✓ RESOLVED At most 3% of slippage is now permitted: The code has furthermore been updated to allow for a variable amount to be withdrawn compared to the full balance.

Issue #10	**Typographical issues and gas optimizations**
Severity	● INFORMATIONAL

Description	Lines 37-39
	
	`if (_stargateNative != address(0) && IOFT(_stargateNative).token() != address(0)) {`
	`  revert InvalidNativeStargate();`
	`}`
	
	This validation is already present within `DonateCore` and could be removed if desired.

Recommendation	Consider fixing the typographical issues and gas optimizations.
Resolution	✓ RESOLVED