PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For LayerZero Aptos ZRO Airdrop

14 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 Aptos ZRO Airdrop on the Ethereum and Aptos 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 Aptos ZRO Airdrop
URL	https://layerzero.network/
Platform	Ethereum and Aptos
Language	Solidity and Move
Preliminary	https://github.com/LayerZero-Labs/monorepo/blob/72c2875b1979-5c61bab8113e38c56393bdb3e757/apps/bridge-v1/contracts-aptos-/airdrop-zro/sources/airdrop-zro.move https://github.com/LayerZero-Labs/monorepo/blob/72c2875b19795c61bab8113e38c56393bdb3e757/apps/bridge-v1/-contracts-evm/contracts/AptosAirdropZRO.sol
Resolution	https://github.com/LayerZero-Labs/monorepo/blob/e01c6dadbe014-f2d32f8afcd0fef95568552cdeb/apps/bridge-v1/contracts-aptos/air-drop-zro/sources/airdrop-zro.move https://github.com/LayerZero-Labs/monorepo/blob/e01c6dadbe014f2d32f8afcd0fef95568552cdeb/apps/bridge-v1/con-tracts-evm/contracts/AptosAirdropZRO.sol

1.2 Contracts Assessed

Name	Contract	Live Code Match
airdrop_zro (Aptos)		PENDING
AptosAirdropZRO (Ethereum)		PENDING

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
Governance	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	4	2	-	2
Informational	5	1	1	3
Total	**9**	**3**	**1**	**5**

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 airdrop_zro (Aptos)

ID	Severity	Summary	Status
1	LOW	Module lacks resource account usage which reduces governance security flexibility	ACKNOWLEDGED
2	LOW	Freely configurable adapter parameters could allow claimers and donation senders to provide too little gas for execution, requiring manual execution	✓ RESOLVED
3	LOW	send_donations_to_guild relies on coin_bridge::send_coin to fail with the testnet destination id, as otherwise coins could be sent to the wrong address and chain	✓ RESOLVED
4	LOW	Airdrops can be claimed to the zero address even though such messages can never be delivered due to the delivery failing	ACKNOWLEDGED
5	INFO	EventHandle-based events are deprecated	ACKNOWLEDGED
6	INFO	Lack of separation between configuration and claiming phase could cause administrative issues	✓ RESOLVED
7	INFO	calculate_donation rounds in favor of the user, which may cause no donation to be levied for very small airdrops.	ACKNOWLEDGED
8	INFO	Typographical issues	ACKNOWLEDGED

1.3.2 AptosAirdropZRO (Ethereum)

ID	Severity	Summary	Status
9	INFO	Typographical issues	PARTIAL

2 Findings

2.1 airdrop_zro (Aptos)

The `airdrop_zro` module is a simple OAPP on the Aptos network. It allows for the oapp owner (the LayerZero team) to assign `LZO` token airdrops to aptos accounts.

Subsequently, these accounts can claim their airdrop. Claiming occurs by bridging a message to the Ethereum mainnet `AptosAirdropZRO` OAPP, which transfers the `LZO` tokens to the receiver. An Ethereum address is specified at the time of claiming by the user. The user is furthermore responsible for covering the LayerZero bridge gas costs of this claim transaction.

An additional charge called the "donation" is levied for claiming: This requires users to pay $0.1 per `LZO` token in either `USDC` or `USDT` on Aptos to claim their tokens.

This donation is kept within the module until anyone calls `send_donations_to_guild`. Once `send_donations_to_guild` is called, the coins will be sent to the configured `MAINNET_GUILD_ADDRESS` address on the Ethereum mainnet, using the LayerZero Aptos bridge. During the resolution round, this guild address has been configured to `0x25941dC771bB64514Fc8abBce970307Fb9d477e9`.

2.1.1 Privileged Functions

- `add_airdrop`
- `cancel_airdrop`
- `set_pause [ INTRODUCED DURING RESOLUTIONS ]`

2.1.2 Issues & Recommendations

Issue #1	Module lacks resource account usage which reduces governance security flexibility
Severity	LOW SEVERITY
Description	The `airdrop_zro` module stores all of its resources within the `signer` global resource store. This has the downside that only the off-chain account authentication methods are supported. Module-based authorization/authentication (eg. a multisig or timelock) is not easily supported as a result, reducing the security flexibility.
Recommendation	Consider using a resource account to store all resources in, and initialize the module with.
Resolution	ACKNOWLEDGED The client wishes to keep changes minimal but will carefully manage the ownership keys of the main account.

Issue #2	**Freely configurable adapter parameters could allow claimers and donation senders to provide too little gas for execution, requiring manual execution**
Severity	LOW SEVERITY
Description	Line 134 `adapter_params: vector<u8>,` Presently, the `claim_airdrop` and `send_donations_to_guild` functions allow for the caller to provide arbitrary `adapter_params`, these parameters communicate options for destination execution. Most notably, the gas parameter can be configured via `adapter_params`. This can be problematic if it's set too low, which would require manual re-execution. This may brick the ordered execution fully until manual execution occurs. This can also be exceptionally annoying if `send_donations_to_guild` is spammed with small donations that all need to be manually executed.
Recommendation	Consider automatically generating the adapter parameters, or simply providing an empty array. It may make sense to still have the actual gas amount be configurable by a privileged account. The `quote` function could furthermore be adjusted as well.
Resolution	RESOLVED The client has fixed this within `claim_airdrop`: The gas parameter is now validated. The client has furthermore indicated that this is already checked within `send_donations_to_guild` as the underlying bridge validates the parameter.

Issue #3	**send_donations_to_guild relies on coin_bridge::send_coin to fail with the testnet destination id, as otherwise coins could be sent to the wrong address and chain**
Severity	LOW SEVERITY

Description	Lines 140-144
	
	`let dst_receiver: vector<u8> = if(dst_eid == ETHEREUM_MAINNET_EID) {`
	`  MAINNET_GUILD_ADDRESS`
	`} else {`
	`  TESTNET_GUILD_ADDRESS`
	`};`
	
	Presently the `send_donations_to_guild` function lacks any form of direct validation to enforce that the donation can only be sent to the mainnet EID in production. Instead, it relies on the endpoint architecture to detect that the testnet eid is not registered there, which is indeed unlikely.
	
	If for some reason this eid is registered however, this could cause the coins to be sent to the wrong network.

Recommendation	Consider validating that the `eid` is configured explicitly via `remote::contains`, or even better consider only permitting the correct `eid` from being used.
Resolution	RESOLVED The client has indicated that these two eids will never be configured at the same time and that they will be careful to keep it that way. They have furthermore indicated that this is indeed already asserted at other levels to fail while unconfigured. No changes were made.

Issue #4	**Airdrops can be claimed to the zero address even though such messages can never be delivered due to the delivery failing**
Severity	LOW SEVERITY
Description	Line 158 `receiver: vector<u8>,` Users are free to provide any EVM address as the receiver of their airdrop, including the zero address. However, when they send it to the zero address, the message will never be able to execute on the destination chain as that transfer will always fail. This can be especially annoying if the LayerZero team builds some auto-retry mechanism for these failed transfers, which would need special exception logic for this case. The non-zero requirement is present within the `Transfer` library used on the destination `OAPP`, alongside with likely being present within the `LZO` token as well. Note that a similar attack is possible to any blacklisted `LZO` recipient. However, there's no way to deal with that except for allowing such messages to fail delivery.
Recommendation	Consider validating that `receiver` is non-zero beforehand.
Resolution	ACKNOWLEDGED

Issue #5	EventHandle-based events are deprecated
Severity	INFORMATIONAL
Description	Line 61 `claim_events: EventHandle<ClaimEvent>` The module employs the `EventHandle` method for emitting events. However, Aptos indicates within their documentation that this methodology has been deprecated for the module events. It should be noted that using `EventHandle` events still completely works, and there is hence no direct downside for not respecting the deprecation from our perspective.
Recommendation	Consider moving to module events.
Resolution	ACKNOWLEDGED

Issue #6	Lack of separation between configuration and claiming phase could cause administrative issues
Severity	INFORMATIONAL
Description	Presently, `claim_airdrop` can be called as soon as an account has received an airdrop allocation via `add_airdrop`. This has the downside that if people figure out what the airdrop module is while the team is still configuring it, they can start claiming airdrops right away, even though configuration hasn't finished. This is especially annoying if one of the `add_airdrop` calls were to fail or be dropped for some reason. In that case it will become very difficult for the team to figure out which accounts need to be retried as they cannot simply rely on the difference between the airdrop amount and the `pending_airdrop` amount.
Recommendation	Consider making the contract pausable and starting in the paused state. At least the `claim_airdrop` function should only be permitted while unpaused, though the `send_donations_to_guild` function could also be protected if desired. Next, it may make sense to only allow for `add_airdrop` to be callable while paused. Finally, we recommend the team to do a careful check at the end of the full configuration by iterating over all accounts and their `pending_airdrop` function, to ensure that no double allocation was given. `add_airdrop` is not idempotent, after all.

Resolution	RESOLVED Claiming is now paused by default, and can be unpaused and re-paused by the team at will.

Issue #7	**calculate_donation rounds in favor of the user, which may cause no donation to be levied for very small airdrops.**
Severity	INFORMATIONAL
Description	Line 205 `airdrop * DONATION_PER_TOKEN / (math64::pow(10, ZRO_DECIMALS) as u256)` The `calculate_donation` function rounds down, meaning that if `airdrop` is exceptionally small, no donation may be required. Generally, rounding in favor of the user can be a risk. However, since here this is an additional cost for the user, there is hardly any risk. The user is furthermore no longer able to escalate the impact of this rounding benefit as partial claims have been disabled in the secondary scope freeze.
Recommendation	Consider rounding against the favor of the user if desired.
Resolution	ACKNOWLEDGED

Issue #8	Typographical issues
Severity	INFORMATIONAL
Description	Line 54

```
struct ClaimEvent has drop, store {
```

This event lacks the donation amount, alongside with the sender address. Contrary to EVM events, we do not believe the signer of the event trigger is trivially accessible off-chain.

Line 64

```
fun init_module(account: &signer) {
```

It may make sense to explicitly validate that the address for `account` is equal to `@airdrop_zro`. This is not typically done within Aptos contracts but would further reduce the possibility for accidental misconfiguration.

Lines 91 and 104

```
assert!(@airdrop_zro == address_of(signer), ENOT_AUTHO-
RISED);
```

These signer validations can be moved above the `airdrop_store` borrow to put the validations above the function body, as is common practice as it results in "failing early" in case the assertions fail.

Lines 94-97

```
if (!smart_table::contains(&airdrop_store.airdrop,
recipient)) {
  smart_table::add(&mut airdrop_store.airdrop, recipient,
0);
};
let current_rewards = smart_table::borrow_mut(&mut aird-
rop_store.airdrop, recipient);
```

The above snippet can be simplified to `smart_table::borrow_mut_with_default`, which combines all of the above lines into a single clean statement.

Line 117

```
msglib_params: vector<u8>,
```

This should not be configurable as it serves no purpose within Aptos LayerZero V1. Consider removing the argument and instead providing an empty vector as the send parameter.

Lines 119 and 137

```
assert!(dst_eid == ARBITRUM_MAINNET_EID || dst_eid ==
BSC_TESTNET_EID, EINVALID_EID);
assert!(dst_eid == ETHEREUM_MAINNET_EID || dst_eid ==
AVALANCE_TESTNET_EID, EINVALID_EID);
```

These checks could be made stricter by enforcing them to equal a specific EID. Though we understand that such specialization would complicate the codebase, as a config existence check already happens within the message lib and remote configuration. Hence no change is strictly required from a security perspective.

Line 169

```
assert!((donation as u256) >= calculate_donation(amount)
, EINVALID_DONATION);
```

We do not understand the benefit of this check not being an equality check.

Recommendation	Consider fixing the typographical issues.
Resolution	● ACKNOWLEDGED

2.2 AptosAirdropZRO (Ethereum)

The `AptopsAirdropZRO` contract is the Ethereum OAPP receiver for the `airdrop_zro` aptos module. It receives "airdrop" claim messages which originated from the Aptos application, by users claiming on Aptos, and distributes the claimed `LZO` tokens to the EVM address provided by the user.

The contract `owner` (the Aptos team) can at any point reclaim all Aptos within this contract, effectively terminating the airdrop. This is presumably present for migration purposes. We've not created an issue for this as the benefits of being able to migrate and finalize the airdrop are valid from our perspective, but this is nonetheless a governance risk.

The `AptopsAirdropZRO` contract is furthermore pausable by the owner. While paused, tokens will not be transferred to recipients and those LayerZero messages will need to be manually retried once the contract is unpaused.

2.2.1 Privileged Functions

- `withdraw`
- `setPause`
- `setConfig`
- `setSendVersion`
- `setReceiveVersion`
- `forceResumeReceive`
- `setTrustedRemote`
- `setTrustedRemoteAddress`
- `setPrecrime`
- `setMinDstGas`
- `setPayloadSizeLimit`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Issues & Recommendations

Issue #9	Typographical issues
Severity	INFORMATIONAL

Description	Line 21
	
	`function withdraw(address _to, uint256 _amount) external onlyOwner {`
	
	It may make sense to also let the `owner` provide the token to withdraw, to also be able to withdraw any other tokens accidentally sent to this contract.
	
	Lines 18, 21, 25 and 29
	
	`trustedRemoteLookup[_aptosEid] = abi.encodePacked(_peer, address(this));`
	`function withdraw(address _to, uint256 _amount) external onlyOwner`
	`function setPause(bool _paused) public onlyOwner`
	`function _nonblockingLzReceive(uint16 _srcEid, bytes memory, uint64, bytes memory _message) internal override`
	
	All of these functions lack events.
	
	Line 40
	
	`to := mload(add(_message, 20))`
	
	Dirty bits might be prepended to this `to` variable. Though to our knowledge solidity always cleans these dirty bits correctly, hence there should not be an issue. If desired, the bits can be cleaned explicitly by wrapping this in a `shr(96, shl(96, ...))` statement.

Recommendation	Consider fixing the typographical issues.
Resolution	PARTIALLY RESOLVED The token address can now be provided within the `withdraw` function. All other typographical issues have been left as-is.