Three Sigma Labs

Code Audit

Metazero

MetaZero Protocol

Gaming RWA Tokenization

Disclaimer

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Table of Contents

Summary

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Summary

Three Sigma Labs audited MetaZero Protocol in a 3 day engagement. The audit was conducted from 27-01-2024 to 30-01-2024.

Protocol Description

MetaZero is a Synthetic Liquidity Layer Protocol for Omnichain Tokenization of Gaming Real World Assets (RWAs), powered by LayerZero.

Scope

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Scope

ONFTV2-721C

Assumptions

Openzeppelin, LayerZero and LimitBreak are trusted.

Methodology

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Project Dashboard

Application Summary

Name	MetaZero Protocol
Commit	a562bc6b73d55ee0059240701e81e938cbe95890
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	27-01-2024 to 30-01-2024
№ of Auditors	2
Review Time	3 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	2	2	0
High	2	2	0
Medium	3	1	2
Low	3	1	2

None	6	5	1

Category Breakdown

Suggestion	6
Documentation	0
Bug	5
Optimization	2
Good Code Practices	3

Code Maturity Evaluation

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Weak. Detokenize and send where not checking the owner or approval of nfts.
Arithmetic	Satisfactory. No arithmetic errors were found.
Centralization	Weak. Minters can freely mint nfts.
Code Stability	Satisfactory. The code was stable throughout the audit.
Upgradeability	Weak. The contracts are not upgradeable.
Function Composition	Satisfactory. The code was correctly split into helper functions.
Front-Running	Satisfactory. No front-running issues are present.
Monitoring	Moderate. Some events were missing.
Specification	Satisfactory. The code follows the specifications.
Testing and Verification	Weak. Some issues should have been found by tests.

Findings

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Findings

3S-MetaZero-C01

NFTs can be stolen by calling **`send()`** and receiving the nfts in another chain

Id	3S-MetaZero-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in f34458d.

Description

send() does not check the owner of the to be sent nfts, which allows anyone to send nfts to itself on behalf of someone else. See the POC here.

Recommendation

Add **`_isApprovedOrOwner()`** on the nfts to be sent in **`send()`**.

3S-MetaZero-C02

`deTokenize()` is missing access control, anyone can burn other people's nfts

Id	3S-MetaZero-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Status	Addressed in 5b68968.

Description

deTokenize() allows users to burn their nfts with a certain string **`url`**. The function is missing access control, such that anyone can burn any nft, POC here.

Recommendation

Either place the **`onlyOwner`** modifier or call **`_isApprovedOrOwner()`**.

3S-MetaZero-H01

Send should revert if the gas limit has not been set for a destination chain (peer)

Id	3S-MetaZero-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in 98f36e9.

Description

send() allows users to send messages with 0 gas limit if the **`gasLimit`** mapping has not be seen for a certain peer. This will likely make the transaction revert on the destination chain, losing the funds. POC here.

Recommendation

Revert if trying to send a message to **`_dstEid`** whose gas limit was not yet set.

3S-MetaZero-H02

Lost nfts due to smart wallets having different addresses on different chains

Id	3S-MetaZero-H02
Classification	High
Severity	Critical
Likelihood	Low
Category	Suggestion
Status	Addressed in 5f138f1.

Description

send() sends the tokens to the same address that called the function. Some smart contract wallets may have different addresses on different chains (or even chains with different address generation).

Recommendation

Let the user specify a destination address to send the nfts to.

3S-MetaZero-M01

`mintGenesis()` mints with a push pattern using **`_mint()`** which may lead to lost tokens

Id	3S-MetaZero-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion
Status	Acknowledged

Description

`_mint()` in mintGenesis() may send tokens to an address that can't deal with ERC721. **`_safeMint()`** would be recommended; however, it would introduce additional issues, namely reentrancy and possible malicious reverts.

Recommendation

Use a merkle tree to allow users to claim their genesis NFTs. This way, each user is responsible for claiming their NFT correctly. Additionally, it would be much cheaper for the owner of the MetaZeroVortex contract and the **`isGenesis`** mapping could be verified against the merkle tree, saving up to 1000 storage writes.

3S-MetaZero-M02

In function **`mint()`** and **`lzReceive()`**, **`_safeMint()`** is recommended over **`_mint()`**, which performs additional checks

Id	3S-MetaZero-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion
Status	Acknowledged

Description

Function mint() and lzReceive() mint a nft to a certain address, which may lead to lost tokens if **`_safeMint()`** is not used instead. This recommendation should not apply to **`mintGenesis()`**, the proposed fixed there is using a Merkle Tree.

Note: if **`lzReceive()`** reverts due to **`_safeMint()`**, the token is already burnt in the source chain, which could mean that the nft is forever lost. However, if this happens, a minter can manually call **`safeMint()`** to the user (or another address specified by the user) in the destination chain. Note that the endpoint's behavior is non blocking now by default, such that if the lzReceive reverts, other users may continue sending cross chain nfts https://docs.layerzero.network/explore/layerzero-v2#improved-message-handling. **`mint()`** (or **`safeMint()`**, if the protocol fixes the usage of **`_mint()`**), should have a flag specifying if the nft is a genesis one, so if **`_safeMint()`** reverts on **`lzReceive()`**, it's still possible to mint the genesis nft. Additionally, if **`lzReceive()`** fails, the owner must clear the payload beforehand in the lzEndpoint manually if it wants to call **`safeMint()`** separately. If a minter mints the lost token to the user before clearing the payload in the LzEndpoint, the user may act maliciously and send the newly minted token to another chain, burning it in the current chain, and then try to execute the payload again in the LzEndpoint, getting a free nft.

3S-MetaZero-M03

Can not mint 1000 tokens, only 999

Id	3S-MetaZero-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in cd1bb46.

Description

Only 999 tokens can be limited according to the code, but the comments indicated that 1000 tokens should be minted on genesis. See the POC for confirmation.

Recommendation

Change genesisCounter to `0`.

3S-MetaZero-L01

Important events are missing

Id	3S-MetaZero-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in a7dab03.

Description

State changing variables should emit events for data fetching purposes. The following variable changes are missing events:

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L55-L56

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L78

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L135

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L163

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L192

https://github.com/threesigmaxyz/metazero-issues-external/blob/master/src/ONFTV2-721C.sol#L196

3S-MetaZero-L02

`renounceOwnership()` should be disabled if it is not intended to be ever used

Id	3S-MetaZero-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Calling **`renounceOwnership()`** would block the owner from very important functions, which would be problematic. When this is the case and the owner is not supposed to be renounced in the future, it's best to disable the **`renounceOwnership()`** function.

3S-MetaZero-L03

`Ownable2Step` should be preferred over `Ownable`

Id	3S-MetaZero-L03
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

Ownable performs a simple ownership transfer by setting the owner to a new address. This is dangerous and could result in an invalid owner being set. Ownable2Step performs additional checks by setting a pending owner which has then to accept the ownership. Consider using it instead.

3S-MetaZero-N01

genesisMint gas costs can be reduced by caching **genesisCounter**

Id	3S-MetaZero-N01
Classification	None
Category	Optimization
Status	Addressed in 6ff3111.

Description

The genesisMint() function could cache **genesisCounter** while looping:

```
uint256 cachedGenesisCounter = genesisCounter;
for (uint i; i < _to.length; ++i) {
    _mint(_to[i], _ids[i]);
    isGenesis[_ids[i]] = true;
    cachedGenesisCounter++;
}
genesisCounter = cachedGenesisCounter;
```

3S-MetaZero-N02

`_setDefaultRoyalty()` in the constructor is overriding the `BasicRoyalties` constructor

Id	3S-MetaZero-N02
Classification	None
Category	Good Code Practices
Status	Addressed in 28850d5.

Description

`_setDefaultRoyalty()` in the constructor is overriding the `BasicRoyalties` constructor to 500, regardless of the parameter sent.

3S-MetaZero-N03

Unused parameters names can be removed to ignore compiler warnings

Id	3S-MetaZero-N03
Classification	None
Category	Good Code Practices
Status	Addressed in d7d5c84.

Description

lzReceive() only uses the **payload** parameter, so the other names can be removed, leaving only the types:

```
function _lzReceive(
    Origin calldata, // struct containing info about the message sender
    bytes32, // global packet identifier
    bytes calldata payload, // encoded message payload being received
    address, // the Executor address.
    bytes calldata // arbitrary data appended by the Executor
```

3S-MetaZero-N04

Hardcoded variables should be placed as constants

Id	3S-MetaZero-N04
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

The LzEndpoint, a minter, the royalty fee and the pay in zero flags ([1], [2]) are hardcoded values, but should be placed as constants instead.

3S-MetaZero-N05

`genesisLimit` can be placed as constant variables to save gas.

Id	3S-MetaZero-N05
Classification	None
Category	Optimization
Status	Addressed in a79fbc8.

Description

Constant variables are cheaper as no storage writes are required, so **`genesisLimit`** should define the **`constant`** keyword.

3S-MetaZero-N06

`pragma abicoder v2;` is enforced for solidity versions above **`0.8.0`**

Id	3S-MetaZero-N06
Classification	None
Category	Suggestion
Status	Addressed in 03b38ab.

Description

Solidity versions above **`0.8.0`** enforce **`pragma abicoder v2;`** by default.