Three Sigma

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 StakingContract in a 3 day engagement. The audit was conducted from 06-04-2024 to 08-04-2024.

Protocol Description

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

Scope

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Scope

StakingContract.sol

Assumptions

OpenZeppelin is secure.

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	dd1a55695d1d757766195cd31b64a5fd7bbd9813
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	06-04-2024 to 08-04-2024
№ of Auditors	2
Review Time	3 days

Vulnerability Summary

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

None	5	2	3

Category Breakdown

Suggestion	2
Documentation	0
Bug	5
Optimization	2
Good Code Practices	1

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	Satisfactory. All access control is correctly implemented.
Arithmetic	Moderate. Some rounding errors were found.
Centralization	Satisfactory. No significant points of centralization are found.
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	Satisfactory. Most events are emitted.
Specification	Satisfactory. The code follows the specifications.
Testing and Verification	Satisfactory. The codebase implements unit and fuzz tests.

Findings

Code Audit

MetaZero Protocol Gaming RWA Tokenization

Findings

3S-MZ-H01

`feesAccrued` or staker principal may be sent as rewards if **`rewardRate`** and **`emissionEnd`** are not properly calculated

Id	3S-MZ-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Status	Addressed in #dffac73.

Description

As **`emissionEnd`** and **`rewardRate`** can be freely set (although **`rewardRate`** only in the **`constructor`**), if **`emissionEnd x rewardRate`** is bigger than the amount of tokens sent to the staking contract, stakers may withdraw **`feesAccrued`** or other stakers' principal as rewards.

Recommendation

Limit the rewards to claim to **`IERC20(basicToken).balanceOf(address(this)) - totalStaked - feesAccrued`**.

To simplify the logic, it might be easier to only unstake in function **`completeUnstake()`**, but leave the rewards stored in **`stakers[msg.sender].reward`** to be claimed by calling **`claimReward()`**.

In **`claimReward()`**, instead of deleting **`staker.rewards`**, set it to **`staker.rewards = staker.rewards - actuallyWithdrawnRewards`**.

3S-MZ-H02

Increasing **emissionEnd** after the previous **emissionEnd** ended will yield full rewards according to **newEmissionEnd - prevEmissionEnd**

Id	3S-MZ-H02
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #7455c9c.

Description

Rewards are emitted at a certain **rewardRate**, which means stakers that have been staking for longer receive more rewards. However, there is an edge case in which this does not happen.

When the **emissionEnd** is reached, **lastApplicableTime()** will always be **emissionEnd**. This means that anyone who stakes after **emissionEnd**, no matter how much time after it, will have the same **rewardDebt** stored.

Thus, if the **emissionEnd** is increased after it has ended, every staker will receive the same rewards according to **newEmissionEnd - prevEmissionEnd**, even if they have staked in the same block that the **emissionEnd** was increased.

Putting this thought into numbers, if **emissionEnd = 10**, and if at **block.timestamp = 15**, a new **emissionEnd** is set to **20**, stakers that staked from the beginning will get rewards pro-rata to **15-10** at the current timestamp, the same as stakers who staked at the current **block.timestamp = 15**, if they frontrun the **setEmissionDetails()** call.

Recommendation

The protocol decided to remove this functionality.

3S-MZ-L01

`getRemainingUnstakeTime()` returns `0` if user has not unstaked

Id	3S-MZ-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

`getRemainingUnstakeTime()` returns `0` if the user has not initiated unstaking.

Recommendation

If `staker.unstakeInitTime == 0` return `unstakeTimeLock`.

3S-MZ-L02

`emissionStart` has no effect, rewards will start accumulating starting on the first staker

Id	3S-MZ-L02
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #67d85d0.

Description

`emissionStart` is set in the constructor, but it has no use other than summing with the **`emissionDuration`** to set the **`emissionEnd`**. Rewards will be accrued as long as **`block.timestamp`** is before **`emissionEnd`** and **`totalStakedAccruingRewards`** is non null.

Recommendation

Either remove this variable or change the logic to accumulate rewards only after emission starts.

3S-MZ-L03

Missing some events

Id	3S-MZ-L03
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Acknowledged

Description

Events in the constructor are not emitted.

`rewardPerToken` update could emit an event.

`withdrawFees()` is missing **`feesAccrued`** event.

`completeUnstake()` is missing fee paid event.

Recommendation

Emit events for all state changes.

3S-MZ-N01

Variables should be cached to memory to save gas

Id	3S-MZ-N01
Classification	None
Category	Optimization
Status	Acknowledged

Description

Storage reads are expensive and should be avoided. This is usually done by caching storage variables in memory ones.

Recommendation

Replace all storage rewards that happen more than once per function (or even better, per flow) for memory caching.

Example:

```
function rewardPerToken() public view returns (uint256) {
    uint256 totalStakedAccruingRewardsCached = totalStakedAccruingRewards;
    if (totalStakedAccruingRewardsCached == 0) {
        return rewardPerTokenStored;
    }
    return rewardPerTokenStored + (
        (lastApplicableTime() - lastUpdateTime) * rewardRate * 1e18 /
totalStakedAccruingRewardsCached
    );
}
```

3S-MZ-N02

Custom errors should be used instead of **`require()`** statements to save gas

Id	3S-MZ-N02
Classification	None
Category	Optimization
Status	Acknowledged

Description

`require()` statements spend more gas than custom errors.

Recommendation

Replace **`require()`** statements with custom errors.

3S-MZ-N03

`completeUnstake()` not following `checks-effects-interactions` pattern

Id	3S-MZ-N03
Classification	None
Category	Suggestion
Status	Addressed in #67d85d0.

Description

`completeUnstake()` transfers tokens before deleting the information of the staker, not following the `checks-effects-interactions` pattern.

Recommendation

Delete the staker `mapping` before transferring the tokens.

3S-MZ-N04

Constants should not be hardcoded

Id	3S-MZ-N04
Classification	None
Category	Good Code Practices
Status	Addressed in #67d85d0.

Description

Constants should not be hardcoded due to readability purposes.

Recommendation

Replace `10000` by `uint256 private constant BASIS_POINTS = 10000`,

`200` by `uint256 private constant MAX_FEE = 200` and

`15 days` by `uint256 private constant MAX_TIMELOCK = 15 days`.

3S-MZ-N05

Codebase is not using **`SafeERC20`**

Id	3S-MZ-N05
Classification	None
Category	Suggestion
Status	Acknowledged

Description

The codebase is using **`transferFrom()`** and **`transfer()`** which revert for weird **`ERC20`** tokens. It should not happen as the MetaZero token implements the standard correctly.

Recommendation

Use SafeERC20::safeTransferFrom() and SafeERC20::safeTransfer() instead.