OpenZeppelin | security

Zeebu Reward Distribution Audit

ZEEBU

September 13, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-08-01 To 2024-08-21
Languages	Solidity
Total Issues	46 (23 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	6 (1 resolved)
Low Severity Issues	17 (9 resolved, 2 partially resolved)
Notes & Additional Information	22 (12 resolved)

Scope

We audited the TechnologyZeebu/OZWaitlistContractsAudit repository at commit d669ac3 and the TechnologyZeebu/OZContractsAudit repository at commit 9c2dfe5.

In scope were the following files:

```
OZWaitlistContractsAudit/
├── rate.sol
├── ReentrancyGuard.sol
├── referral.sol
└── UserWaitList.sol

OZContractsAudit/contracts/
├── Launchpad.vy
├── Lens.sol
├── RewardDistributor.sol
└── VotingEscrow.vy
```

System Overview

The Zeebu protocol consists of two isolated parts intended to be deployed on the BASE network: the reward distribution system and the referral points system. These modules are not interconnected and work independently with their own objectives.

Reward Distributor

The `RewardDistributor` contract is the main entry point for users. It is designed to allow accounts with the `REWARD_DISTRIBUTOR_ROLE` role to add rewards to be distributed to all `veToken` holders in the `VotingEscrow` contract. Rewards can be in any token and each distribution is defined by an `amt` (amount) of a custom `token` and the `rewardTimestamp`, tracked by a mapping. When adding a new reward distribution, the `amt` of `tokens` is transferred into the `RewardDistributor` contract and remains there for users to claim. Users can claim these rewards by calling the `claim` function while providing an array of distribution IDs anytime after their `rewardTimestamp` time has passed. Calculations for the value for each user are based on the user's `veToken` balance, relative to the total supply at the time of reward distribution in the `VotingEscrow` contract. This means that the system will distribute rewards proportionally to the users.

The `VotingEscrow` system is based on Curve's `VotingEscrow` contract but with a start timestamp for each locked position. This timestamp is used to calculate the early withdrawal penalty based on the time passed compared to the total locked time. This is different from the Curve implementation which is based on the `MAXTIME` value. Curve's `VotingEscrow` system is designed to allow users to lock their governance tokens in exchange for voting power (`veToken`). The amount of `veTokens` minted back to the user depends on the amount of governance tokens locked and the locked period length: the longer the period, the more `veTokens` are minted, with a maximum locking period of 4 years. Over time, as the lock period decreases, the amount of `veToken` decays linearly, reducing the user’s voting power. This encourages long-term commitment to the protocol. The `VotingEscrow` contract is also able to calculate a user's voting power in any given timeframe which is then used by Zeebu's `RewardDistributor` contract as a snapshot of token balance when calculating and distributing rewards to all `veToken` holders.

Referral Point System

This system consists of two main contracts: the `ReferralCodeMapping` contract and the `UserWaitList` contract.

The `ReferralCodeMapping` contract implements functionality that keeps track of referral codes (limited to `10**10`) against their underlying referrer and the linkage of parents and children. Some of this functionality is only accessible to privileged entities, one of whom is the `UserWaitList` contract. Even though the process of linking a user to a referral code consists of a single step, the process for linking parents and a child, on the other hand, consists of registration and then acceptance. Logic in the `ReferralCodeMapping` contract converts the referral code to a string, and to do so, it uses the `StringUtils` library and other similar functionalities present in the contract.

The `UserWaitList` contract is the actual entry point for users. It keeps track of the accumulated points after keeping certain checkpoints on user profiles. Points are awarded by registering a user, activating the referral, and validating the email (an admin action). The admin can also then redeem the points for rewards which are kept in a separate mapping. However, no assets are transferred in this process. The contract contains an accounting system that is complementary to the one in the `ReferralCodeMapping` contract. The users can be suspended in this contract which would prevent them from accumulating any more points or converting them into rewards.

Trust Assumptions and Privileged Roles

The system has many privileged roles with various powers. In general, these roles are EOAs controlled by Zeebu and are trusted not to abuse their power. Some of the privileged actions that users having these roles can perform are listed below.

In the `UserWaitList` contract, the allowed users can:

- Define the amount of rewards a user will receive from the accumulated points, even though the user might be entitled to more rewards.
- Set the `userType` of a user.

In the `UserWaitList` contract, the `admin` user can:

- Perform all the actions that the allowed users can.
- Initiate the contract.
- Validate the email of a user.
- Suspend/unsuspend users.
- Set the internal variables of the contract, such as the maximum reward cap for users, the token to use for the balance check, and the `ReferralCodeMapping` contract's address.
- Add/remove whitelisted users.
- Withdraw any token from the `UserWaitList` contract.

In the `ReferralCodeMapping` contract, the `admin` can:

- Add/remove whitelisted users.

In the `ReferralCodeMapping` contract, the whitelisted users can:

- Add a referral code to a user.
- Add a child to a parent.
- Accept the child to a previously added parent.

In the `RewardDistributor` contract, the `REWARD_DISTRIBUTOR_ROLE` role can:

- Add a single or multiple distributions.

In the `tokensUSDRate` contract, the `UPDATE_ROLE` role can:

- Set the new rate for any token.
- A malicious permissioned user could DoS a rate update by sending a higher `operationNonce` and front-running another rate update. This will revert the honest rate update as it will have a nonce that is up to 1000 units smaller.

In the `VotingEscrow` contract, the `admin` role can manage the contract settings including:

- The smart contract checker
- The `admin` role
- The early unlock and penalty settings
- All unlock settings
- The reward receiver address

While there are other participants in the protocol that matter in the accounting system, they do not affect the on-chain flow directly. These are OLP, DEPLOYER, VALIDATOR, and DELEGATOR.

Moreover, there are flows that users must be aware of, such as:

- A highly restrictive over-removal of distribution because the current time must be after the distribution takes place and the `totalSupply` has become zero. This could result in problems such as a distributor accidentally adding a reward far into the future that cannot be removed, removing the rewards halfway through the rewards period, and removing unclaimed rewards after the rewards claim has settled.
- Due to the nature of the reward calculations in the `RewardDistributor` contract, the last user to claim the reward might potentially receive more due to rounding errors.

Generally, all privileged roles are expected to be available at any time for emergency actions and are expected to monitor all external integrations for good behavior (e.g., ensuring that all swaps performed by the protocol are through the optimal swap path). We encourage the Zeebu team to put in place automated monitoring to help detect emergencies, and to track the information coming from the `tokensUSDRate` contract for correctness. It is also recommended to track all the point/reward/fund movements to identify potential malicious behaviors. We also encourage the Zeebu team to define all the known sources of emergencies which could result in suspending users or changing parameters in different contracts. This should be followed by monitoring for these sources on-chain by utilizing tools such as Forta or OpenZeppelin Defender.

Considerations and Potential Improvements

Throughout the audit, several opportunities for improvement were identified:

- The codebase includes several contracts from the OpenZeppelin contracts library that have been pasted directly into other files. It is recommended to use a package management system to import these dependencies. This will help ensure that the official release versions are used and no unintentional changes are made.
- Throughout the codebase, there was a lack of documentation which, in many cases, makes it difficult to understand the intention of a given functionality.

- Test coverage was limited. We recommend aiming for 100% coverage of the codebase to ensure that all functionality has been properly tested.
- There were several instances in the codebase where the logic and code structure could be improved by failing early to save gas and implementing more efficient variable packing.
- There are features that are currently unused and appear to be placeholders for future use. These could be removed to simplify the codebase.
- The overall file structure for both repositories does not follow the standard structure used by smart contract developers. In particular, contracts should be in their own directory and only configuration and documentation should be in the root directory.
- There is a lack of documentation regarding how some on-chain values will be used by off-chain systems related to user fund allocation.
- Code that has been forked from other protocols still contains references to the original project and retains unused functionality.

Addressing the above concerns will ensure a high-quality and maintainable codebase, benefiting future developers and auditors alike.

Update: *The fixes for the identified issues were provided in bulk rather than as individual pull requests for each issue, as originally requested. Due to this submission format, the verification of each fix may not be fully comprehensive. While we have made every effort to review the implemented changes, the bulk nature of the submission could affect the precision of our review. Our focus was placed on verifying the two higher-severity issues (H01 and M06), along with the remaining 22 issues that were addressed. Additional code changes were introduced beyond those directly related to the issues. These additional changes were not fully reviewed, as our review was limited to the specific issue-related modifications.*

High Severity

H-01 Reward Time Window is Calculated Incorrectly

The `rewardTime` variable defines the cooldown period. It is calculated as `minute * hour * day * _rewardTime` where `minute == 1` and the other two constants are linked to that variable, multiplied by the input from the `initializeRewardConfiguration`. However, since the outcome is then compared with `block.timestamp` (denominated in seconds), two problems arise:

- As the `_rewardTime` input from the `initializeRewardConfiguration` function is supposedly denominated in seconds, the `rewardTime` outcome will be shorter by a factor of 60. This is because `minute` corresponds to 1 and not 60.
- Each unit of time is calculated using its previous smaller units, but the calculation for getting the `rewardTime` multiplies them once again.

Consider adjusting the `minute` constant to 60 (or using another `second` constant) and only using the `day` constant (which should hold the number of seconds in a day from the chain of multiplications). Doing this will correctly represent the calculation of the `rewardTime` variable and reduce the likelihood of accidentally having an off-cooldown period.

Update: *Resolved at commit 6bd78de. The code was updated to only use DAYs in seconds to multiply the given `rewardTime` which should be in days.*

Medium Severity

M-01 Rewards Can Be Easily Farmed by Bots

The design of the referral system is not robust enough and can be easily farmed by bots. In particular, a malicious user could use:

- Bots along with any registered user as a referrer and then register to get rewards.
- Bots to easily activate the referral account for extra rewards.

- The same balance of tokens can be used to bypass the registration process and register multiple users at the same time. This is because that balance has not been locked in an epoch. As such, bots can transfer the same balance from account(i) to account(i+1) after calling the `registerUser` function.

Consider implementing functions like token balance snapshots to counter bot activities.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on the business requirements.*

M-02 Multiple Sources of Truth Design

System design should favor a single source of truth (SSOT). However, the codebase does the opposite. For example, `referral.sol` tracks referrals with the `userToReferralCount` function while `waitlist.sol` also tracks the same thing using the `referralUserStatus` function in line 577. Multiple sources of truth make the system less efficient and more vulnerable to issues and attacks.

Consider updating the system design to have a single source of truth.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *The referral contract is used exclusively by the `Waitlist` contract.*

M-03 User Rewards Could Get Slashed

The `UserWaitList` contract implements the functionality to reward users in exchange for registering on the network. Users get a bonus for each step they perform while the protocol caps that accumulated reward up to the `maxRewardPoints` parameter. However, the admin could call the `setMaxRewards` function and pass a lower value. Consequently, if a user were in the middle of a process that changes their reward balance (e.g., activating or verifying the email), the user could get their accumulated reward balance slashed. This will happen if the accumulated points become lesser than the new `maxRewardPoints` value even though they were earned before the new cap.

To prevent users' funds from being slashed when a new cap is set, consider checking if the current balance is bigger than the new cap in any of the processes that alter the balance. If so, account for the surplus balance of the respective user in the `redeemRewards` mapping.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on business requirements and the admin will ensure that the values are updated accordingly.*

M-04 `UserWaitList` Can Be Initialized Multiple Times

The `UserWaitList.initializeRewardConfiguration` function does not have any "check now" modifier that prevents the `admin` role from calling it multiple times, even after it has already been configured and users have been interacting with the contract. Since other setter functions are meant to be called afterwards, like the one for the `maxRewardPoints` variable, they should only be allowed to be set once by using an initializable contract or some similar implementation. Likewise, the `initializeUserPoolLimitConfiguration` function also lacks a check to prevent multiple calls by the `admin` without checking the previous value. This also means that to avoid unfavorable values, a user might see the `admin`'s call in the mempool and front-run that transaction to get a better deal.

Consider either locking the configuration after it has been initialized or providing a clear explanation for allowing this behavior.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on business requirements and we have changed the method name to `updateRewardConfiguration`.*

M-05 DoS Upon Referral Activation

After a user has been registered into the `UserWaitList` contract, the next step is to activate the their referral, which registers the user in the `ReferralCodeMapping` contract with an input code. However, a malicious user could create a contract that encapsulates the calls to `registerUser` and `activateReferral`, wait for another user to call `activateReferral` with a particular `code`, and then front-run that transaction. This will end up reverting the honest user's call due to a check for an empty value.

To prevent the aforementioned scenario from happening, consider using the registration counter as the code to use the range sequentially.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on business requirements. Additionally, only the allowed contract can call `referralMapping`.*

M-06 Staked Value for `UserTypes` Does Not Account for Changing Types

The `UserWaitList` contract implements different user types that can be used in parallel with the reward system. To set a particular type, the allowed user must call the respective setter, passing in the address of the user and the staked amount which will be recorded in the internal storage. However, none of these functions takes into account the possibility that the same user might change its own type, in which case the "pool status" value in the old type will not reflect reality as it was only added and never subtracted. Moreover, the counters for each type reflect the same behavior and do not show the actual number of users associated with that type once users start to change types.

Consider implementing functionality that checks if a user belongs to a previous type, and fix the accounting system to reflect the real values of staking amount and type counters.

Update: *Resolved at commit 6bd78de. The `UserTypes` setter function has been removed.*

Low Severity

L-01 Incomplete and missing Docstrings

Throughout the codebase, there are several instances of incomplete and missing docstrings. For example functions in `UserWiatList.sol`, `referral.sol`, `RewardDistributor.sol`, `Lens.sol`.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #1. The Zeebu team added docstrings for certain functions in `RewardDistributor.sol`.*

L-02 Require Statement With Multiple Conditions

Throughout the codebase, there are `require` statements that require multiple conditions to be satisfied. For instance:

- The `require(signerExists[signer2] && signerExists[signer3],'Invalid signer')` in `UserWaitList.sol`.
- The `require(signer2 != signer3 && signer1 != signer2 && signer1 != signer3, 'Invalid signer')` in `UserWaitList.sol`.
- The `require(operationNonce > lastOperationNonce && operationNonce <= (lastOperationNonce+1000), 'Enter Valid operationNonce')` in `UserWaitList.sol`.
- The `require(operationNonce > lastOperationNonce && operationNonce <= (lastOperationNonce+1000), 'Enter Valid operationNonce')` in `rate.sol`.

To simplify the codebase and raise the most helpful error messages for failing `require` statements, consider having a single require statement per condition.

Update: *Resolved at commit 6bd78de.*

L-03 Lack of Validation

Throughout the codebase, several instances of unvalidated contract addresses were identified. This means that it is not ensured that the address indeed belongs to the appropriate contract and/or there is code at this address. For instance, the addresses for the `UserWaitList` and `ReferralCodeMapping` contracts are not validated.

To avoid accidentally passing an incorrect address, consider verifying that the contract address being linked contains code. For a more robust solution, consider using an introspection standard.

Update: *Resolved at commit 6bd78de.*

L-04 Missing Storage Gaps in `RewardDistributor`

The `RewardDistributor` contract is an upgradable contract. However, it does not have any storage gaps. This might be problematic if a future upgrade wants to extend and add storage to this contract.

Consider adding storage gaps to the `RewardDistributor` contract.

Update: *Resolved in pull request #1.*

L-05 Unidirectional Linkage Allows Having a Child as a Parent

The `addChild` function from the `ReferralCodeMapping` contract allows passing an already linked parent <> child, but backwards. This is due to the fact that the `parentChildStatus` mapping links in a unidirectional way (parent->child) but not the other way around for the same mapping. This makes it possible to set the parent as the child of the child and vice versa. While the actual place where the function is used does prevent this from happening by requiring the child to not be registered, the `addChild` function should nonetheless perform the check. Moreover, the contract would allow another allowed address to cause such effect when calling the `ReferralCodeMapping` contract directly.

Consider adding the child registration check to the `addChild` function.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *The referral contract is used exclusively by the `Waitlist` contract.*

L-06 Refactoring Opportunities

Throughout the codebase, multiple instances of refactoring opportunities were identified:

- The `StringUtils` library used by the `UserWaitList` and the `ReferralCodeMapping` contracts could be moved into a different file and imported by both contracts.
- The `uintToStringInternal` and `codeToString` functions have been implemented twice in the `UserWaitList` and `ReferralCodeMapping` contracts. Instead, both of

these functions can be moved to the `StringUtils` library from where any contract can use them.

- The `codeToString` function in `referral.sol` takes the `referralCode` (a `uint`) and converts it to a string using an internal function. However, for this conversion, the value used is the code that has already been converted to a string in the `addUserWithReferralCode` function. This means that both strings are equal, which results in the contract redundantly using the passed string's length and data to check the converted string's length and data. This is not efficient as both of the strings are equal, making the second conversion unnecessary.
- The `setRateForToken` function should validate the nonce before the rest of the address recovery. This because the nonce is being used in address recovery as well.

Consider implementing the aforementioned refactoring suggestions to improve code clarity and efficiency.

Update: *Resolved at commit 6bd78de.*

L-07 External Dependencies Copied Into the Codebase

Throughout the codebase, multiple instances of external dependencies having been copied into the local files were identified. Particularly in `UserWaitList.sol` and `rate.sol`. Doing this could cause potential incompatibility between versions while making the code less readable and difficult to maintain. If a particular dependency is needed locally (e.g., when altering part of the functionality that cannot be overridden with the regular import flow), it is better to keep it as one contract/library per file and then import it when needed instead of flattening the contract. Furthermore, there are places at which custom implementations could be replaced with one of the contracts from the OpenZeppelin Contracts library (e.g., the custom ownership in the `ReferralCodeMapping` contract).

Consider using OpenZeppelin contracts where possible instead of copying contracts locally into the code. Alternatively, consider using one modified contract/library per file and documenting all the changes made compared to the original version.

Update: *Resolved at commit 6bd78de.*

L-08 Implementation Inconsistency

The `uintToStringInternal` and `uintToString` functions implement similar functionality but are not completely identical. There are castings that do not show up in the analogous function and the conditions for `while` do not match in both cases. In addition, the `onlyAllowed` modifier from the `UserWaitList` contract differs from the one present in the `ReferralCodeMapping` contract, the former contains the admin in the allowed role whereas the latter does not.

In order to improve code readability and maintainability, consider implementing identical functionalities in a consistent manner.

Update: *Resolved at commit 6bd78de.*

L-09 Validation of 10-Digit Code Is Not Implemented Everywhere

In the `ReferralCodeMapping` contract, when using the `codeToString` function, if the `_referralCode` is bigger than 10 decimal digits, the zero padding will not be congruent with the limitation. Although there is a validation for having a length of 10 after the function has been used, the `getUserAddressByReferralCode` function using the same underlying functionality does not have the same check. Given that the `getUserAddressByReferralCode` function is used by the `UserWaitList` contract to verify the reference code, it should be also validated that it does not exceed the limit and reverts as the other function does.

Consider implementing validation to not accept codes greater than 10 digits in all accessible functions.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It will only return an address. If any string does not match, a null address will be returned.*

L-10 Missing Functionality to Remove or Cancel a Child

In the `ReferralCodeMapping` contract, a child address can be added and accepted but cannot be removed or cancelled. In addition, each new child address is pushed into an array, with no way of removing an address from it, making the array only larger with time. This increases the possibility of a DoS if the array's size gets too big and the array is later used by another contract to loop over the child addresses. Furthermore, children that are added but not accepted cannot be removed from the same array even though they will not count towards the active children for the parent.

Consider implementing functionality to remove children from a father and mitigate a possible DoS vector for contracts using the `getChildAddresses` function.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on the business requirements.*

L-11 Accounting Could Be Inconsistent due to Changes in the Registration Token

The registration token can only be changed by the admin. However, the `token` global variable is updated each time someone wants to register and it takes the current value of the registration token for this purpose. This means that if the registration token is changed during the life cycle, not all the referrals will share the same token. In addition, a malicious user could front-run the call to the `setRegistrationToken` function in case the old token had a potential issue/flaw. Since the registration token is used to check against the minimum value for `ZeeBuTokens` tokens, it should not be allowed to change. Moreover, if the admin does not have enough `ZeebuTokens` tokens, they could change the address to a malicious token with enough balance, call the referral registration, and then reset the token address back to the correct one.

Consider disallowing changes to the registration token address once the protocol becomes operational.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on the business requirements.*

L-12 Users' Array Could Cause a DoS

In the `UserWaitList` contract, whenever a user is registered, its address is pushed into the `usersList` array. This information is used only by the `getUsersList` function which is in charge of returning the entire array. However, as the list keeps growing in size, if any other contract calls this function to retrieve the list and then continues with the execution, it may reach a point at which the large size of the list could cause a DoS in the execution.

Consider only using the `getUsersList` function for external viewing purposes and documenting the aforementioned potential issue for other developers. Alternatively, consider removing the function entirely.

Update: *Partially resolved at commit 6bd78de.*

L-13 Rate Signer is Immutable

The `adminSigner` from the `tokensUSDRate` contract is immutable, meaning that in the event that its private key is lost or compromised, the protocol will not have the power to update the signer and with that it will not be able to set the rate due to the check on the `setRateForToken` function.

Consider allowing the admin of the contract the possibility to change the signer in case something goes wrong.

Update: *Resolved at commit 6bd78de.*

L-14 Unnecessary Storage Pointer

The `addChild` function in the `ReferralCodeMapping` contract uses a a storage pointer for referrals. However, this is unnecessary since nothing is changed on that reference and it is only used to read data from it.

Consider using a `memory` pointer instead.

Update: *Resolved at commit 6bd78de.*

L-15 Child Can Be On Multiple Parents

In the `ReferralCodeMapping` contract, an allowed user can add a child user to an parent address. Part of the validation done is over the existence of the parent address as an added user, that the child is not the parent address nor the zero one, that the child has not accepted the parent, and that the relationship between both from the parent standpoint is null.

However, as the there is no cross validation from the child point of view, an allowed user can add the same child to different parents if those has not accepted the child yet. This is due to the lack of a change in the `childrenToParent` mapping when adding a child. Moreover, all affected parents will get the child address pushed into their array and it will be able to be removed from the no-accepted parents. Also, the `inActive` value will always increase and it will be able to come back to zero. That means that the check in 128 will not have any weight on the affected parents due to the offset in the count.

Furthermore, as the `UserWaitList` contract is only one of the allowed users, any other allowed address (even a EOA) could interfere with the validations done in the mentioned contract, breaking the assumptions and the flow on the correct behavior.

In order to keep the right amount of counts and relationship between the users, consider closing the loop back from the child to the parent to prevent having a single child with multiple parents.

Update: *Resolved at commit 6bd78de.*

L-16 VotingEscrow Cannot Handle Distributions Properly

The `RewardDistributor` contract is heavily connected to the `VotingEscrow` contract, as it is being used to track the voting power of users, the total supply, and allow the `VotingEscrow` contract to add a distribution.

However, the `VotingEscrow` contract is incapable of handling any of the other methods implemented in the `RewardDistributor` contract, in particular the functionalities to add multiple distributions at the same time and removing any kind of distribution made from the `VotingEscrow` contract. This means that in the scenario that a distribution made from it needs to be taken from the protocol, the implementation will fail as the one calling for the removal needs to be the one that placed the distribution (in this case, the `VotingEscrow` contract).

In order to prevent wrongly placed distributions from not being able to be removed and to allow the protocol to have the plain capacity of its features, consider adding the respective functionalities in the `VotingEscrow` contract.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *This is not necessary based on the system design.*

L-17 Insufficient test coverage

Several concerns regarding the testing of the current system which pose a high systemic risk were identified during the audit.

Insufficient testing, while not a specific vulnerability, implies the high probability of additional unfound vulnerabilities and bugs. It also exacerbates multiple interrelated risk factors in a complex code base. This includes a lack of full implicit specification of the functionality and exact expected behaviors that tests normally provide, which increases the chances that correctness issues will be missed. It also requires more effort to establish basic correctness and reduces effort spent exploring edge cases, thereby increasing the chances of missing complex issues.

Moreover, the lack of repeated automated testing of the full specification increases the chances of introducing breaking changes and new vulnerabilities. This applies to both previously audited code and future changes to currently audited code. Underspecified interfaces and assumptions increase the risk of subtle integration issues, which testing could reduce by enforcing an exhaustive specification.

To address these issues, we recommend implementing a comprehensive multi-level test suite. Such a test suite should comprise of contract-level tests with 95%-100% coverage, per chain / layer deployment and integration tests that test the deployment scripts as well as the system as a whole, and per chain / layer fork tests for planned upgrades. Crucially, the test suite should be documented in a way that a reviewer can set up and run all these test layers independently of the development team. Implementing such a test suite should be a very high priority to ensure the system's robustness and reduce the risk of vulnerabilities and bugs.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *We tried to cover all possible test cases.*

Notes & Additional Information

N-01 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

Throughout the codebase, instances of revert and/or require messages were found, for example in `UserWaitList.sol`, `referral.sol`, `rate.sol`.

For conciseness and gas savings, consider replacing require and revert messages with custom errors.

Update: *Resolved at commit 6bd78de.*

N-02 Using `int/uint` Instead of `int256/uint256`

Throughout the codebase, there are uses of `int/uint`, as opposed to `int256/uint256`. For instance:

- In line 978 of `UserWaitList.sol`.
- In line 994 of `UserWaitList.sol`.
- In line 1069 of `UserWaitList.sol`.
- In line 1077 of `UserWaitList.sol`.
- In line 805 of `rate.sol`.
- In line 813 of `rate.sol`.
- In line 821 of `rate.sol`.

Consider replacing all instances of `int/uint` with `int256/uint256`.

Update: *Resolved at commit 6bd78de.*

N-03 Multiple Contract Declarations Per File

Throughout the codebase, there are instances of more than one contract, library, or interface declaration in the same file. For instance:

- The `UserWaitList.sol`.
- The `rate.sol`.
- The `referral.sol`.

Consider separating the contracts into their own files to make the codebase easier to understand for developers and reviewers.

Update: *Resolved at commit 6bd78de.*

N-04 Use Calldata Instead of Memory

When dealing with function parameters in external functions, it's more efficient to use `calldata` instead of `memory`. `calldata` is a read-only region of memory that contains the function arguments, which makes it cheaper and more efficient compared to `memory`. Using `calldata` can save gas and improve the performance of your smart contract.

Within `UserWaitList.sol`, the `_email` parameter should use `calldata` instead of `memory`.

Consider using `calldata` for function parameters in external functions to optimize gas usage.

Update: *Resolved at commit 6bd78de.*

N-05 Lack of Indexed Event Parameters

Throughout the codebase, several events do not have indexed parameters, for example:

- The `UserSuspended` event of `UserWaitList.sol`.
- The `UserUnsuspended` event of `UserWaitList.sol`.
- The `UserRegistered` event of `UserWaitList.sol`.
- The `ReferralCodeActivated` event of `UserWaitList.sol`.
- The `RewardRedeemed` event of `UserWaitList.sol`.
- The `ReceivedTokens` event of `UserWaitList.sol`.
- The `ReferralEmailVerified` event of `UserWaitList.sol`.

- The `userPoolConfigurationUpdate` event of `UserWaitList.sol`.
- The `withdrawalToAdmin` event of `UserWaitList.sol`.

And more.

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *Indexing events increases gas costs, so we have not indexed the parameters. Additionally, we have an off-chain implementation to retrieve the actual parameter values from the events.*

N-06 Variables Are Initialized to Their Default Values

Throughout the codebase, some variables are being initialized to their default values. For instance:

- The `_finalRefCode` variable in `UserWaitList.sol`.
- The `_finalRefCode` variable in `referral.sol`.

To avoid wasting gas, consider not initializing variables to the same values they would have by default when they are being declared.

Update: *Resolved at commit 6bd78de.*

N-07 Constants Not Using `UPPER_CASE` Format

In `UserWaitList.sol` there are constants not declared using `UPPER_CASE` format. For instance:

- The `minute` constant declared in line 626.
- The `hour` constant declared in line 627.
- The `day` constant declared in line 628.

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved at commit 6bd78de. The Zeebu team stated:*

> *It no longer exists as we have updated it to the Day constant.*

N-08 Function Visibility Overly Permissive

Throughout the codebase, there are various functions with unnecessarily permissive visibility, to name a few:

- The `validateWithdrawSigner` function in `UserWaitList.sol` with `internal` visibility could be limited to `private`.
- The `getChainId` function in `UserWaitList.sol` with `internal` visibility could be limited to `private`.

and more.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider doing a overall check and changing a function's visibility to be only as permissive as required.

Update: *Resolved at commit 6bd78de.*

N-09 Inconsistent Capwords Style

Throughout the codebase, there are multiple components that are not following the Solidity style guide. For instance:

- The `whiteListUpdate` event should be named in CapWords style, such as `WhiteListUpdate`.
- The `contractMapUpdate` event should be named in CapWords style, such as `ContractMapUpdate`.
- The `configParamUpdate` event should be named in CapWords style, such as `ConfigParamUpdate`.
- The `rewardConfigurationUpdate` event should be named in CapWords style, such as `RewardConfigurationUpdate`.
- The `userPoolConfigurationUpdate` event should be named in CapWords style, such as `UserPoolConfigurationUpdate`.
- The `withdrawalToAdmin` event should be named in CapWords style, such as `WithdrawalToAdmin`.
- The `tokensUSDRate` contract should be named in CapWords style, such as `TokensUSDRate`.

- The `rateUpdated` event should be named in CapWords style, such as `RateUpdated`.
- The `referralCount` struct should be named in CapWords style, such as `ReferralCount`.
- The `referralAdded` event should be named in CapWords style, such as `ReferralAdded`.
- The `referralAddedChild` event should be named in CapWords style, such as `ReferralAddedChild`.
- The `referralChildActive` event should be named in CapWords style, such as `ReferralChildActive`.
- The `whiteListUpdate` event should be named in CapWords style, such as `WhiteListUpdate`.
- The `rightsUpdated` event should be named in CapWords style, such as `RightsUpdated`.

To improve the project's overall legibility, consider adhering to the Solidity style guide by naming the contracts, structs, enums, events, and errors using the CapWords style.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged. We will take care of this in the future.*

N-10 Inconsistent Order Within Contracts

Throughout the codebase, there are multiple contracts that deviate from the Solidity Style Guide due to having inconsistent ordering of functions:

- The `UserWaitList` contract in `UserWaitList.sol`.
- The `ReferralCodeMapping` contract in `referral.sol`.

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of functions).

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged, we will take care of this in the future.*

N-11 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType`

`ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Throughout the codebase, there are multiple mappings without named parameters:

- The `signerExists` state variable in the `UserWaitList` contract.
- The `users` state variable in the `UserWaitList` contract.
- The `rewards` state variable in the `UserWaitList` contract.
- The `redeemRewards` state variable in the `UserWaitList` contract.
- The `whitelist` state variable in the `UserWaitList` contract.
- The `referralUserStatus` state variable in the `UserWaitList` contract.
- The `_indexes` state variable in the `EnumerableSet` contract.
- The `_roles` state variable in the `AccessControl` contract.
- The `tokenRateInUSD` state variable in the `tokensUSDRate` contract.
- The `functionExecutionTime` state variable in the `tokensUSDRate` contract.
- The `referralCodeToUser` state variable in the `ReferralCodeMapping` contract.
- The `UserToReferralCode` state variable in the `ReferralCodeMapping` contract.
- The `isUserAdded` state variable in the `ReferralCodeMapping` contract.
- The `parentChildStatus` state variable in the `ReferralCodeMapping` contract.
- The `parentToChildren` state variable in the `ReferralCodeMapping` contract.
- The `childrenToParent` state variable in the `ReferralCodeMapping` contract.
- The `whitelist` state variable in the `ReferralCodeMapping` contract.
- The `userToReferralCount` state variable in the `ReferralCodeMapping` contract.
- The `_isClaimedRewardDistribution` state variable in the `RewardDistributor` contract.

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *We will take care of this in the future.*

N-12 State Variable Visibility Not Explicitly Declared

Within `UserWaitList.sol`, there are state variables that lack an explicitly declared visibility. For instance:

- The `minute` state variable.

- The `hour` state variable.
- The `day` state variable.

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Resolved at commit 6bd78de. The Zeebu team stated:*

> *It no longer exists as we have updated it to the Day constant.*

N-13 Non-explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally-defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity file or when inheritance chains are long.

Throughout the codebase, global imports are being used:

- The import "./ReentrancyGuard.sol"; import in `UserWaitList.sol`.
- The import "./ECDSALIB.sol"; import in `UserWaitList.sol`.
- The import "./ECDSALIB.sol"; import in `rate.sol`.

Following the principle that clearer code is better code, consider using the named import syntax *(`import {A, B, C} from "X"`)* to explicitly declare which contracts are being imported.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged. We will take care of this in the future.*

N-14 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, contracts do not have a security contact.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged. We will take care of this in the future*

N-15 Unused Event

In `referral.sol`, the `rightsUpdated` event is unused.

To improve the overall clarity, intentionality, and readability of the codebase, consider emitting or removing any currently unused events.

Update: *Resolved at commit 6bd78de.*

N-16 Inconsistent Use of Named Returns

To improve the readability of the codebase, the same return style should be used.

Within the `RewardDistributor` contract, the `claim`, `claimable`, `_claimable` functions use named return variables whereas the `rewardDistributionsLength` function does not.

Consider being consistent with the use of named returns throughout the codebase.

Update: *Acknowledged, not resolved.*

N-17 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- In lines 909, 919, and 929 of `UserWaitList.sol`, `stackAmount` should be `stakeAmount`.
- In line 654 of `UserWaitList.sol`, `must uniq` should be `must be unique`.

Consider fixing the aforementioned typographical errors to improve code readability.

Update: *Resolved at commit 6bd78de.*

N-18 Styling Errors

Throughout the codebase, multiple instances of styling errors were identified:

- This unneeded space between the mapping's name and the brackets.
- This missing space before `_email`.
- This missing space after the `=`.

To improve code consistency and clarity, consider fixing the aforementioned styling errors.

Update: *Resolved at commit 6bd78de.*

N-19 Functions Could Fail Earlier

Throughout the codebase, some functions could be refactored in favour of failing early and saving gas. In particular:

For `UserWaitList.sol`

- Line 735 and Line 736 could go before Line 732,
- Line 794 could be before Line 793.

For `Rates.sol`

- Line 772 could be placed before Line 771.

For `RewardDistributor.sol`

- Line 213 could be placed before Line 200.

Consider refactoring these functions to improve the codebase.

Update: *Resolved in pull request #1 at commit 6bd78de.*

N-20 Magic Numbers in the Code

Throughout the codebase, there are occurrences of literal values with unexplained meanings.

For instance:

- The `2` literal number in `UserWaitList.sol`.
- The `1000` literal number in `UserWaitList.sol`.
- The `1000` literal number in `rate.sol`.

Consider defining and using `constant` variables instead of using literals to improve the readability of the codebase.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged. We will take care of this in the future.*

N-21 Misleading Naming

When adding a child user with the `addChild` function from the `ReferralCodeMapping` contract, the `referralCount` struct for the parent user is updated and the `inActive` counter adds one.

However, the `inActive` counter does not actually represent the accepted number, which is when the state becomes active, but the pending ones.

This could cause some confusion in both the code's readability and the query by other contracts. Consider changing this variable to `inPending` to reflect how many have not been accepted yet.

Update: *Acknowledged, not resolved. The Zeebu team stated:*

> *It is based on business requirements.*

N-22 Variables Could Be Tightly Packed

Throughout the codebase, multiple instances of variables that can be tightly packed were identified:

- The `User` struct in line 581 can be packed better (e.g., `referrer` and `usertype` can be packed together)
- Some state variables can be of a smaller type for tight packing (e.g., `rewardTime` could be of type `uint32`).

Consider tightly packing the aforementioned variables to reduce storage and gas costs.

Update: *Acknowledged, will resolve. The Zeebu team stated:*

> *Acknowledged. We will take care of this in the future.*

Conclusion

The reward distribution system of the Zeebu protocol consists of contracts that are meant to be deployed on the BASE network. These contracts handle many pieces of functionality related to reward allocation. The audited codebase includes the changes made to the voting escrow used to get the voting power weight for the reward calculation, the launchpad which deploys the contracts, the reward distribution mechanism which is in charge of adding and claiming the rewards, and the referral functionalities which accumulate points that can be exchanged for future rewards. Most of the sensitive functionality is locked behind administrative roles.

Generally, the code under review could benefit from some improvements in order to enhance its security. Some features have been inherited from the Curve codebase, whereas others have been implemented in such a way that they are either unreachable during execution or are disjointed. As such, it is recommended that any unused functionality be removed from the codebase for the sake of simplicity and to ensure that the protocol behaves as expected in the future. In addition, forking other protocols may result in problems when integrating with them. Thus, forking should be approached carefully and with deep knowledge of the inner workings and security risks of both protocols. Furthermore, different contract languages were used for the protocol which could potentially result in the introduction of bugs.

During the audit, we observed that some areas of the codebase needed further attention. This includes updating the documentation, ensuring that the accounting systems for the various contracts are correctly applied in all critical parts of the codebase, and making sure that the calculations for the reward mechanism are straightforward and accurate. Also, a stronger testing suite is necessary to make the codebase, in general, more robust. In order to increase user trust and confidence in the protocol, it is important that more documentation is provided regarding all the moving parts of the codebase and that more importance is placed on improving the code quality.

The Zeebu team has a significant degree of control over the audited contracts which places a big responsibility on their shoulders since it requires the users to trust the protocol and have faith in the development team. Hence, we strongly encourage the Zeebu team to be cognizant of this fact and expand the test suite by including fuzz testing along with integration testing of the imported contracts. Doing this adds intentionality to the development process and reduces the chances of a severe vulnerability going unnoticed. Moreover, if major changes are made before deployment, a new round of auditing would be advisable, apart from internal audits and strict adherence to the Solidity/Vyper Style Guide.

We appreciate the Zeebu team's diligent responses to the audit team's questions, along with their forthcomingness to share any requested information. Both fast communication and timely code delivery are necessary to take the best out of the assigned auditing time.