Zunami v2 Reduced Scope Review

BROKEN WITH ❤ BY NOMOI

We reviewed the https://github.com/ZunamiProtocol/ZunamiProtocolV2 repository at commit 50c825e.

The review started on *Monday, October 23, 2023*.

This report was updated on *Wednesday, November 8, 2023*.

Introduction

The focus of our security assessment centered on a subset of the Zunami v2 protocol contracts. This targeted review aimed to independently evaluate the project's smart contract security, code quality, and operational robustness.

The Zunami protocol issues aggregated collateralized stablecoins. Users deposit collateral which is invested into different strategies provided and managed by the Zunami DAO. The only strategies in scope were the ones related to USD stablecoins.

General recommendations

The Zunami team has demonstrated a commendable effort in striving to meet tight deadlines and has been cooperative throughout the audit process.

However, our review revealed a considerable reliance on contract inheritance within the codebase, presenting challenges for external reviewers in understanding the logic. The absence of documentation, inline comments and NatSpec in the revised protocol, alongside an incomplete test suite, necessitated an extensive focus on interpreting the intended functionality. This combination limited our capacity to fully evaluate the code's robustness within the allocated audit period.

In light of these observations and findings, we advise addressing the issues identified within this report, and we also strongly suggest a subsequent, full-system audit before launching the protocol, once exhaustive documentation and tests are finished.

Scope

The smart contracts in scope for this review were:

```
.
├── ZunamiPool.sol
├── ZunamiPoolCompoundController.sol
├── ZunamiPoolControllerBase.sol
├── ZunamiPoolThroughController.sol
├── configs
│   ├── aps
│   │   ├── ZunamiPooApsUZD.sol
│   │   ├── ZunamiPooApsZETH.sol
│   │   ├── ZunamiPooControllerApsUZD.sol
│   │   └── ZunamiPooControllerApsZETH.sol
│   └── omnipool
│       ├── ZunamiPooControllerUZD.sol
│       ├── ZunamiPooControllerZETH.sol
│       ├── ZunamiPooUZD.sol
│       └── ZunamiPooZETH.sol
├── interfaces
│   ├── ICurvePool.sol
│   ├── ICurvePool2.sol
│   ├── ICurvePool2Native.sol
│   ├── ICurvePoolPricable.sol
│   ├── INativeConverter.sol
│   ├── IPool.sol
│   ├── IRewardManager.sol
│   ├── IStableConverter.sol
│   ├── IStrategy.sol
│   └── IWETH.sol
├── strategies
│   ├── VaultStrat.sol
│   ├── ZunamiPoolOwnable.sol
│   ├── ZunamiStratBase.sol
│   └── curve
│       ├── CurveStratBase.sol
│       ├── convex
│       │   ├── ConvexCurveStratBase.sol
│       │   └── interfaces
│       │       ├── IConvexBooster.sol
│       │       ├── IConvexMinter.sol
│       │       └── IConvexRewards.sol
│       └── stakeDao
│            ├── StakeDaoCurveStratBase.sol
│            ├── crvUsdToken
│            │   ├── CrvUsdStakeDaoCurveStratBase.sol
│            │   ├── usdc
│            │   │   └── UsdcCrvUsdStakeDaoCurve.sol
│            │   └── usdt
│            │       └── UsdtCrvUsdStakeDaoCurve.sol
│            └── interfaces
│                ├── IStakeDaoGauge.sol
│                └── IStakeDaoVault.sol
└── utils
    ├── Constants.sol
    ├── FraxEthNativeConverter.sol
    ├── StableConverter.sol
    └── rewardManager
        ├── SellingCurveRewardManager.sol
        ├── SellingCurveRewardManagerFrxEth.sol
        └── interfaces
            ├── AggregatorV2V3Interface.sol
            └── ICurveExchangePool.sol
```

Findings

1. `FraxEthNativeConverter.handle` does not unwrap weth

LIKELIHOOD HIGH IMPACT MEDIUM

The `FraxEthNativeConverter.handle` calls the `unwrapWETH` function if `buyToken == true` (when converting ETH to frxETH). However, instead of unwrapping `amount`, the parameter passed to the function is the return variable `tokenAmount`, which is always 0 at this step. This means that all calls to `handle` with `buyToken == true` will likely revert when attempting to perform the trade through the `fraxEthPool`.

Recommendation

Fix this issue by correctly passing `amount` to the `unwrapWETH` function, and consider thoroughly testing all contracts.

Update: *As of commit `804a885` this issue has been resolved.*

2. `VaultStrat.transferAllTokensTo` only works if the pool uses 5 tokens

LIKELIHOOD HIGH IMPACT MEDIUM

The `transferAllTokensTo` function iterates through the pool's tokens but doesn't stop if one of the tokens is the zero address. This will result in a revert.

Recommendation

Consider breaking from the loop if the token is the zero address, like in the `transferPortionTokensTo` function.

***Update**: As of commit `804a885` this issue has been resolved.*

3. `SellingCurveRewardManager.checkSlippage` assumes `feeToken` decimals

LIKELIHOOD LOW IMPACT MEDIUM

The `SellingCurveRewardManager.checkSlippage` function assumes `feeToken` decimals are always 6, even though it allows an arbitrary `feeToken` as a parameter. This once again could result in unexpected issues if the pool is misconfigured.

Recommendation

Consider explicitly checking if the provided `feeToken` is supported by the reward manager.

***Update**: As of commit `804a885` this issue has been resolved by performing the slippage check with the received `usdtAmount` and delegating slippage checks for the `feeToken` conversion to the `stableConverter` contract.*

4. `StableConverter` functions don't validate their inputs

LIKELIHOOD LOW IMPACT MEDIUM

If an address that is not explicitly used as a key for the `curve3PoolStableIndex` mapping is passed as the `to` or `from` arguments to the `StableConverter` functions , the returned index will be `0` , which is a valid index and could lead to unexpected behavior if the pool is misconfigured.

Recommendation

Consider explicitly checking if the provided tokens are supported by the converter contract.

***Update**: As of commit 28926db this issue has been resolved.*

5. Existing pool tokens are not removed when setting new tokens

LIKELIHOOD LOW IMPACT MEDIUM

The `ZunamiPool._addTokens` function iterates through the provided `tokens_` array and overwrites the values in the `_tokens` array. However, if the new token array is "shorter" than the tokens already in the state, the old tokens won't be cleared so unexpected issues might arise. The controller could actually pass a token array with empty tokens at the end in order to prevent these issues, however this is not guaranteed and might go unnoticed when operating a pool.

Recommendation

Consider always clearing the old token array when setting new pool tokens.

***Update**: As of commit 804a885 this issue has been resolved.*

6. Unprotected functions can lead to reentrancy attacks

LIKELIHOOD LOW IMPACT MEDIUM

The codebase contains several cases in which state changes happen after external calls to other contracts, such as the `ZunamiPoolCompoundController`'s `autoCompoundAll` and `claimManagementFee` functions. These are unprotected functions that can be called by anyone, opening the possibility for reentrancies to happen.

Recommendation

Consider using reentrancy guards on all user facing functions.

***Update**: As of commit `804a885` this issue has been resolved.*

7. `withdrawPercent` is incorrectly validated

LIKELIHOOD LOW IMPACT LOW

The `ZunamiPool._moveFunds` function does not validate that the `withdrawPercent` parameter is lower or equal than `FUNDS_DENOMINATOR`, allowing for invalid `withdrawPercent` values.

Recommendation

Consider reverting if `withdrawPercent > FUNDS_DENOMINATOR`.

***Update**: As of commit `804a885` this issue has been resolved.*

8. Incorrect revert condition can lead to inconsistent state

LIKELIHOOD LOW IMPACT LOW

The `ZunamiPool._addTokens` function reverts if the following condition holds:

```
tokens_.length != _tokenDecimalMultipliers.length && tokens_.length >
POOL_ASSETS
```

However, this condition should use `||` instead of `&&`. One potential issue is that it is possible to set more than `POOL_ASSETS` tokens.

Recommendation

Consider fixing the revert condition.

***Update**: As of commit `804a885` this issue has been resolved by fixing the revert condition.*

9. Incorrect usage of `UpdatedToken` event `tokenOld` parameter

LIKELIHOOD LOW IMPACT LOW

The `ZunamiPool._addToken` function emits the `UpdatedToken` event, always setting the `tokenOld` parameter to `address(0)`, even if a token already exists at the corresponding array index.

Recommendation

Consider checking if a token already exists, and correctly pass it to the `UpdatedToken` event emission.

***Update**: As of commit `804a885` this issue has been resolved.*

10. Lack of input validation in `SellingCurveRewardManagerFrxEth.handle`

LIKELIHOOD LOW IMPACT LOW

The `SellingCurveRewardManagerFrxEth.handle` function does not use the `feeToken` parameter, as it assumes it is `frxETH`. This defeats the purpose of using a generic `IRewardManager` interface, as a user of this interface might not be compatible with this specific reward manager and might pass a different `feeToken`, resulting in unexpected issues.

Recommendation

Consider explicitly checking that `feeToken` equals `frxETH`.

***Update**: As of commit `804a885` this issue has been resolved by reverting if the `feeToken` is not `frxETH`.*

11. Strategy status not always checked

LIKELIHOOD LOW IMPACT LOW

Functions such as `ZunamiPoolControllerBase.setDefaultDepositSid` and `ZunamiPoolControllerBase.setDefaultWithdrawSid` don't check the status of the provided strategy, which could prevent the use of certain controller functions temporarily.

Recommendation

Consider always checking the strategy statuses when these are provided as inputs.

***Update**: Won't fix, intended behavior.*

12. `calcTokenAmount` does not need to be part of the IStrategy interface

ENHANCEMENT

`calcTokenAmount` is only used internally by the `CurveStratBase.calcLiquidityTokenAmount`, and the `VaultStrat.calcTokenAmount` implementation is empty, which might indicate it would be better if this function was inlined whenever needed, instead of providing it through the interface.

***Update**: Won't fix. Justification: function used by the UI.*

13. Improve virtual function implementations

ENHANCEMENT

The implementation of `depositPool` and `withdrawPool` in `ZunamiPoolControllerBase` appears to be designed for extension rather than direct use. This pattern can hinder code comprehensibility due to the need for tracing function logic across multiple levels of inheritance. Specifically, unused parameters, such as the first argument in `withdrawPool`, are symptomatic of this design choice and may lead to confusion or the introduction of errors.

Consider refactoring the codebase to ensure that virtual functions are self-contained and logical in isolation. By doing so, maintainability and readability could significantly improve, easing future audits and development.

***Update**: As of commit `804a885` this issue has been partially resolved by not implementing the `withdrawPool` and `depositPool` functions in `ZunamiPoolControllerBase`.*

14. Lack of validation for component compatibility

ENHANCEMENT

The current design of the protocol attempts to make all components in it (pools, controllers, reward managers, tokens, etc) decoupled between them, by using interfaces. While this makes the protocol flexible in certain aspects, it might also introduce potential issues as not all implementations are compatible with each other. This, combined with the lack of explicit checks for compatibility between components could potentially lead to configuration issues during the operation of the protocol.

A clear example of this is the compatibility between the controller's reward tokens and the `IRewardManager` implementations. Currently, each reward manager only supports trading between specific tokens. However, when setting the controller's reward tokens, it is not checked wether the reward manager actually supports these tokens.

Another example is the compatibility between a specific strategy and the corresponding pool. The current system does not validate that the strategy being used actually matches in any form the pool's configuration, even though the strategies internally use the pool's token indexes, resulting in implicit requirements that might result in unexpected issues if the pool is incorrectly configured. As there are no compatibility checks, the responsibility of preventing these issues is completely delegated to the governance process.

Recommendation

Consider reviewing the design of all interactions between the different components and try to minimize the possibility of compatibility issues.

For the first example mentioned earlier, one option would be to make the `IRewardManager` interface provide an `isTokenSupported` function and explicitly check if each reward token is supported.

15. Magic Values

ENHANCEMENT

Throughout the codebase, there are many cases in which hardcoded values are used within functions, which are not documented. This makes the code harder to reason about and decreases the maintainability of the codebase.

Some examples of this issue:

- `CurveStratBase` : line 76
- `VaultStrat` : line 86
- `ZunamiStratBase` : lines 23, 29, 47, 75
- `ZunamiPoolCompoundController` : line 147
- `SellingCurveRewardManagerFrxEth` : lines 130, 135

To improve the readability of the codebase, consider reviewing the whole codebase and declaring constants with descriptive names (or reusing already declared ones) for each of these magic values.

***Update**: As of commit `804a885` this issue has been resolved.*

16. Naming issues

ENHANCEMENT

The are several cases throughout the codebase where naming of variables and functions could be improved. Using unclear naming hinders the readability and maintainability of the project.

- `ZunamiPoolCompoundController` defines the `managementFee` and `managementFees` variables. It is hard to understand what each of these are supposed to represent without looking at how they are used, and can potentially lead to issues if the developer mistakes one for the other.
- `IPool.StrategyInfo.deposited` should probably be named `minted`, as it tracks the total shares minted by the pool instead of the actual funds deposited in the strategy.
- There are many functions that have unclear names. Some examples of this:
    - `convertLiquidityTokenAmount`: this function receives token amounts and returns a `uint256[2]` array with the equivalent amount of the corresponding zunami tokens, because its output is expected to be passed to curve's pool `calc_token_amount`. None of this is clear from the naming.
    - `calcLiquidityTokenAmount`: this function is named almost the same as the previous one, but its functionality is completely different, as it returns the shares that are expected to be received/deposited.
- `ZunamiPool` `addTokens` and `_addTokens` function should be renamed to `setTokens`, as they completely override existing tokens.

Consider reviewing all function and variable names, and use names that explicitly describe what each function is doing.

***Update**: As of commit `804a885` this issue has been partially resolved.*

17. Prefer explicit setters over toggles

ENHANCEMENT

The `ZunamiPool.toggleStrategyStatus` toggles the status of the provided strategy. This operation is not idempotent and might make operation of the pool more difficult in certain situations, such as when multiple DAO operations are being performed simultaneously.

Consider changing the function to `setStrategyStatus(uint256, bool)`.

***Update**: As of commit `804a885` this issue has been resolved by introducing functions to set a strategy status explicitly.*

18. Prefer IERC20 over IERC20Metadata

ENHANCEMENT

Throughout the codebase, `IERC20Metadata` is used when access to ERC20 functions is needed. However, unless metadata functions such as `symbol`, `name` and `decimals` are needed, using `IERC20` is sufficient and makes the code easier to read.

Update*: As of commit `804a885` this issue has been resolved by replacing `IERC20Metadata` with `IERC20`.*

19. Unnecessary external call to `calcTokenAmount`

ENHANCEMENT

The `CurveStratBase.calcLiquidityTokenAmount` function performs an external call to itself with `this.calcTokenAmount(...)`. This is unnecessary if `calcTokenAmount` is declared as public, and incurs in additional gas costs.

Update*: As of commit `804a885` this issue has been resolved.*