ZUNAMI SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

APRIL 15, 2024

CONTENTS

1 INTRO

OXORIO

1.1 DISCLAIMER

The audit makes no assertions or warranties about the utility of the code, its security, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other statements about the fitness of the contracts for their intended purposes, or their bug-free status. The audit documentation is for discussion purposes only.

1.2 ABOUT OXORIO

Oxorio is a prominent audit and consulting firm in the blockchain industry, offering top-tier security audits and consulting to organizations worldwide. The company's expertise stems from its active involvement in designing and deploying multiple blockchain projects, wherein it developed and analyzed smart contracts.

With a team of more than six dedicated blockchain specialists, Oxorio maintains a strong commitment to excellence and client satisfaction. Its contributions to several blockchain projects reflect the company's innovation and influence in the industry. Oxorio's comprehensive approach and deep blockchain understanding make it a trusted partner for organizations in the sector.

Contact details:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

1.3 SECURITY ASSESSMENT METHODOLOGY

Several auditors work on this audit, each independently checking the provided source code according to the security assessment methodology described below:

1. Project architecture review

The source code is manually reviewed to find errors and bugs.

2. Code check against known vulnerabilities list

The code is verified against a constantly updated list of known vulnerabilities maintained by the company.

3. Security model architecture and structure check

The project documentation is reviewed and compared with the code, including examining the comments and other technical papers.

4. Cross-check of results by different auditors

The project is typically reviewed by more than two auditors. This is followed by a mutual cross-check process of the audit results.

5. Report consolidation

The audited report is consolidated from multiple auditors.

6. Re-audit of new editions

After the client has reviewed and fixed the issues, these are double-checked. The results are included in a new version of the audit.

7. Final audit report publication

The final audit version is provided to the client and also published on the company's official website.

1.4 FINDINGS CLASSIFICATION

1.4.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

- **CRITICAL**: A bug that could lead to asset theft, inaccessible locked funds, or any other fund loss due to unauthorized party transfers.
- **MAJOR**: A bug that could cause a contract failure, with recovery possible only through manual modification of the contract state or replacement.
- **WARNING**: A bug that could break the intended contract logic or expose it to DDoS attacks.
- **INFO**: A minor issue or recommendation reported to or acknowledged by the client's team.

1.4.2 Status Level Reference

Based on the client team's feedback regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

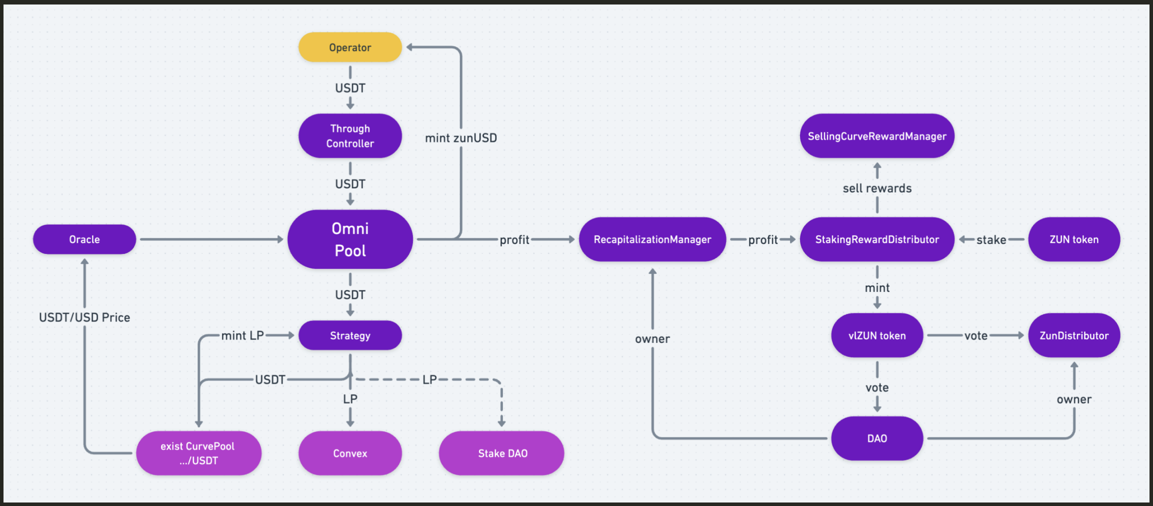
- **NEW**: Awaiting feedback from the project team.
- **FIXED**: The recommended fixes have been applied to the project code, and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Fixes for this finding are planned. This finding does not affect the overall security of the project.
- **NO ISSUE**: The finding does not affect the overall security of the project and does not violate its operational logic.

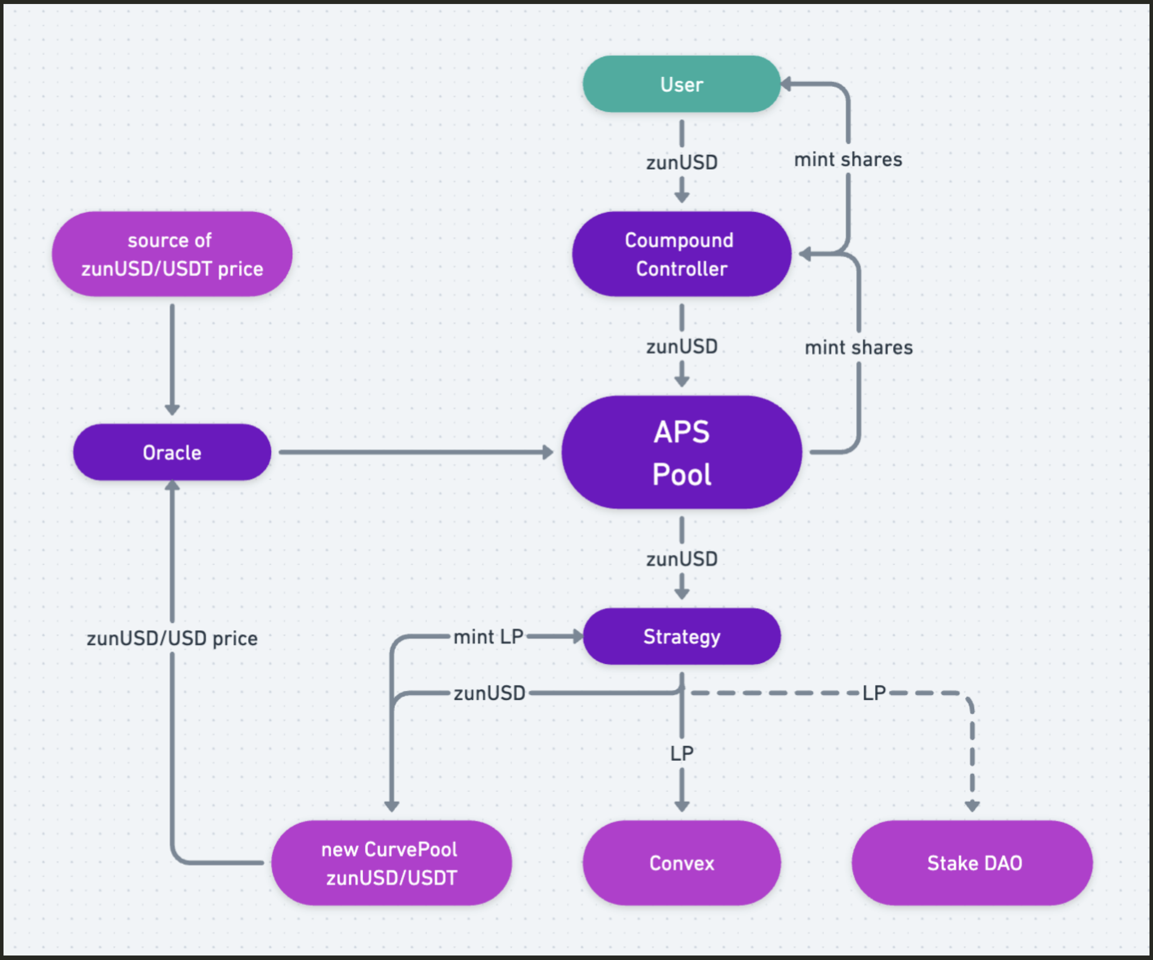
1.5 PROJECT OVERVIEW

Zunami is a protocol that aggregates stablecoin collateral to generate optimized yields.

Users deposit collateral to establish an omnipool. zunStables, representing shares of the omnipool, are issued to users. The omnipool provides liquidity across diverse yield-generating strategies. Profits are distributed among ZUN token stakers, who also hold governance rights. Stakers can vote to approve recapitalization.

System diagrams:





1.5.1 Documentation

For this audit, the following sources of truth about how the smart contracts should work were used:

- main GitHub repository of the project.

The sources were considered to be the specification. In the case of discrepancies with the actual code behavior, consultations were held directly with the client team.

1.6 AUDIT SCOPE

The audit scope covers all smart contracts located in the contracts folder of the project repository.

The audited commit identifier is df33b7c2789b6090932d9b71097b98c103d87329.

The commit identifier with fixes is e16201cb5f594431c122b3c92c9206523c515abe.

2 FINDINGS REPORT

OXORIO

2.1 CRITICAL

C-01	Staked `ZUN` unavailable for withdrawal after `vlZUN` ownership change in `ZUNStakingRewardDistributor`
Severity	CRITICAL
Status	• FIXED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `withdraw`	162

Description

The `ZUNStakingRewardDistributor` contract allows users to `deposit` and `withdraw` funds.

In the `deposit` function the user sends `ZUN` tokens and receives an equivalent of `vlZUN` tokens. The contract storage `userLocks` mapping is also updated to save information about the user's deposit.

```
function _deposit(uint256 _amount, address _receiver) internal {
    //...
    uint128 untilBlock = uint128(block.number + BLOCKS_IN_4_MONTHS);
    uint256 lockIndex = userLocks[_receiver].length;
    userLocks[_receiver].push(LockInfo(uint128(_amount), untilBlock));
    emit Deposited(_receiver, lockIndex, _amount, untilBlock);
}
```

For the `withdraw` function, a user (`msg.sender`) seeking to withdraw staked `ZUN` tokens must have a balance in the `userLocks` mapping and `vlZUN` tokens.

```
function withdraw(
    uint256 _lockIndex,
    bool _claimRewards,
    address _tokenReceiver
) external nonReentrant {
    LockInfo[] storage locks = userLocks[msg.sender];
    if (locks.length <= _lockIndex) revert LockDoesNotExist();

    LockInfo storage lock = locks[_lockIndex];
    if (untilBlock == 0) revert Unlocked();
    uint256 untilBlock = lock.untilBlock;
    uint256 amount = lock.amount

    //...
}
```

When a `vlZUN` token is transferred to a new owner, the staked `ZUN` tokens associated with the previous owner will not be available for withdrawal. This is because the `userLocks` mapping, which tracks locked `ZUN` tokens for each user, is not updated when `vlZUN` tokens are transferred.

This creates a scenario:

- Alice transfers `vlZUN` tokens to Bob.
- Bob holds the `vlZUN` tokens, but Alice keeps the lock information in the `userLocks` mapping.
- Bob can not withdraw the staked `ZUN` associated with the transferred `vlZUN` tokens as the `userLocks` mapping for Bob's address is empty.

Recommendation

We recommend considering two potential solutions. The first option involves a complete removal of the transfer logic from the `vlZUN` token. Alternatively, the second option entails implementing the logic to update the `userLocks` mapping during the `vlZUN` token transfer process.

Update

Client's response

Fixed in commit `dbe84378589a1d4ce13e5ee2bc0a44d2907aed1b`

2.2 MAJOR

M-01	`_updateDistribution` function is vulnerable to change of the contract balance in `BaseStakingRewardDistributor`
Severity	MAJOR
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_updateDistribution`	195
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_updateDistribution`	197

Description

In mentioned cases, the `_updateDistribution` function uses a subtraction assignment operation to assign the result to a variable of type `uint256`. This operation relies on the assumption that the contract balance will never fall below the sum stored in the variables `totalAmount` and `rewardInfo.balance`.

```
function _updateDistribution(
    uint256 _tid,
    uint256 _totalSupply
) internal returns (uint256 distribution) {
    RewardTokenInfo storage rewardInfo = rewardTokenInfo[_tid];
    address token_ = address(rewardInfo.token);
    uint256 dI = 0;
    if (_totalSupply != 0) {
        uint256 tokenBalance = IERC20(token_).balanceOf(address(this));
        if (token_ == address(token)) {

            tokenBalance = _reduceByStakedAmount(tokenBalance);
        }
        dI = (1e18 * (tokenBalance - rewardInfo.balance)) / _totalSupply;
        rewardInfo.balance = tokenBalance;
    }

    distribution = rewardInfo.distribution + dI;
    if (dI != 0) {
        rewardInfo.distribution = distribution;
        emit DistributionUpdated(_tid, distribution);
    }
}
```

```
function _reduceByStakedAmount(
    uint256 _tokenBalance
) internal view virtual returns (uint256 reducedTokenBalance) {
    reducedTokenBalance = _tokenBalance - totalAmount;
}
```

However, due to minor rounding errors or the use of the `withdrawStuckToken` function, the contract balance may fall below the value stored in the storage. In such a situation, the subtraction operation will fail.

As a result of the problem described above, all methods that call the `_updateDistribution` function internally will fail. These methods include `deposit`, `withdraw`, `distribute`, `claim`, and `transfer`.

Example scenario:

- Alice deposits 100 tokens for staking.
- `totalAmount` = 100, `rewardInfo.balance` = 100.
- `withdrawStuckToken` call diminishes the contract balance to 90.
- The `_updateDistribution` function attempts to calculate `dI` using `tokenBalance - rewardInfo.balance`.
- Subtraction fails due to `tokenBalance` being slightly less than `rewardInfo.balance`.
- The function crashes, halting any further distribution of rewards.

Recommendation

We recommend ensuring that the first term in the subtraction expression always exceeds the second. An alternative solution is to allow the distribution difference `dI` to take negative values by using a signed variable.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

M-02	Rewards lost if contract's balance insufficient in `BaseStakingRewardDistributor`
Severity	MAJOR
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointReward`	172

Description

In the `_checkpointReward` function of the `BaseStakingRewardDistributor` contract, when the contract balance is insufficient to pay the entire reward amount, the user only receives a portion of `totalClaimable`. However, the full amount is deducted from the `claimableRewards` variable in the storage.

```
function _checkpointReward(
        uint256 _tid,
        address _user,
        address _receiver,
        uint256 _userBalance,
        uint256 _totalSupply,
        bool _claim
    ) internal {

    //...

    uint256 totalClaimable = claimableRewards[_tid][_user] + newClaimable;
    if (totalClaimable > 0) {
        if (_claim) {
            _safeRewardTransfer(rewardTokenInfo[_tid].token, _receiver, totalClaimable);
            rewardTokenInfo[_tid].balance -= totalClaimable;
            // update amount claimed
            claimedRewards[_tid][_user] += totalClaimable;
            claimableRewards[_tid][_user] = 0;
            emit Claimed(_receiver, _tid, totalClaimable);
        } else if (newClaimable > 0) {
            // update total_claimable
            claimableRewards[_tid][_user] = totalClaimable;
        }
    }
}
```

This behavior results in the user receiving only a partial reward and forfeiting the ability to claim the remaining balance later.

Example scenario:

- Alice accumulates rewards `userDistribution` in the `BaseStakingRewardDistributor` contract.
- The contract calculates the total `totalClaimable` for the user, which includes both the previously accumulated rewards and any newly distributed rewards.
- If Alice attempts to claim the rewards, the contract checks the contract balance. If the contract balance is insufficient to pay the entire `totalClaimable` amount, Alice only receives a partial reward through the `_safeRewardTransfer` function.
- The full `totalClaimable` amount is deducted from Alice's `claimableRewards` variable, even though she didn't receive the full amount.

Recommendation

We recommend reducing the `claimableRewards` storage counter to reflect only the amount of tokens actually transferred to the user, not the entire amount.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

Oxorio's response

In the `_checkpointReward` function of the `BaseStakingRewardDistributor` contract, rewards are transferred to the user's address if they set the `_claim` parameter to `true`, and the user's `claimableRewards` mapping value is updated accordingly:

```
uint256 totalClaimable = claimableRewards[_tid][_user] + newClaimable;
if (totalClaimable > 0) {
    if (_claim) {
        uint256 transferred = _safeRewardTransfer(
            rewardTokenInfo[_tid].token,
            _receiver,
            totalClaimable
        );
        rewardTokenInfo[_tid].balance -= transferred;
        // update amount claimed
        claimedRewards[_tid][_user] += transferred;
        if (claimableRewards[_tid][_user] != 0) {
            claimableRewards[_tid][_user] -= transferred;
    }
    // ...
```

However, this implementation of `claimableRewards` update is not suitable for some edge cases.
If `transferred` > `claimableRewards`:

- `claimableRewards[_tid][_user]` equals `100`
- `newClaimable` equals `100`
- `transferred` equals `150`
- An underflow revert occurs when trying to execute `claimableRewards[_tid][_user] -= transferred;`

If `transferred` < `claimableRewards` then:

- `claimableRewards[_tid][_user]` equals `100`
- `newClaimable` equals `100`
- `transferred` equals `50`
- `claimableRewards[_tid][_user]` is incorrectly set to `50` when it should be `150`

We recommend handling the above edge cases in the following way:

```
if (claimableRewards[_tid][_user] != 0 || totalClaimable > transferred) {
    claimableRewards[_tid][_user] = totalClaimable - transferred;
}
```

2.3 WARNING

W-01	Recapitalization manager must return more than taken in `ZUNStakingRewardDistributor`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `_deposit`	139

Description

In the `_deposit` function of the `ZUNStakingRewardDistributor` contract, if recapitalization occurs before the deposit, the `recapitalizedAmount` variable is proportionally increased when the deposit operation is executed.

```
function _deposit(uint256 _amount, address _receiver) internal {
    //...
    uint256 ratio = getRecapitalizationRatio();
    if (ratio < RATION_DENOMINATOR) {
        uint256 amountReduced = (_amount * ratio) / RATION_DENOMINATOR;
        recapitalizedAmount += _amount - amountReduced;
    //...
}
```

This feature leads to a situation where the recapitalization manager is required to return a number of tokens that exceeds the number actually taken earlier.

Recommendation

We recommend removing the logic that increases the `recapitalizedAmount` variable in the `deposit` function.

W-02	Outdated reward calculation in `BaseStakingRewardDistributor`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `getPendingReward`	249

Description

The `getPendingReward` function in the `BaseStakingRewardDistributor` contract may return an outdated value because it does not consider the current contract balance when calculating the reward.

```
function getPendingReward(uint256 _tid, address _user) external view returns (uint256) {
    uint256 newClaimable = 0;
    if (userRewardDistribution[_user][_tid] < rewardTokenInfo[_tid].distribution) {
        newClaimable =
            (balanceOf(_user) *
                (rewardTokenInfo[_tid].distribution - userRewardDistribution[_user][_tid])) /
            1e18;
    }

    return claimableRewards[_tid][_user] + newClaimable;
}
```

Recommendation

We recommend using logic similar to the `_updateDistribution` call to accurately reflect the current token balance in the contract.

W-03	Missing `rewardInfo.balance` storage value update in `BaseStakingRewardDistributor`
Severity	WARNING
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `withdrawStuckToken`	268

Description

In the `withdrawStuckToken` function of the `BaseStakingRewardDistributor` contract, the `rewardInfo.balance` storage value is not updated if the withdrawn token is one of the reward tokens.

```
function withdrawStuckToken(
    IERC20 _token,
    uint256 _amount
) external onlyRole(DEFAULT_ADMIN_ROLE) {
    uint256 withdrawAmount = _amount == type(uint256).max
        ? _token.balanceOf(address(this))
        : _amount;
    if (withdrawAmount > 0) {
        _token.safeTransfer(_msgSender(), withdrawAmount);
    }
}
```

This leads to incorrect calculations in the `_updateDistribution` function, since the value of `rewardInfo.balance` does not correspond to the current reward balance in the contract.

```
function _updateDistribution(
    // ...
    dI = (1e18 * (tokenBalance - rewardInfo.balance)) / _totalSupply;
    rewardInfo.balance = tokenBalance;
    // ...
}
```

Recommendation

We recommend updating the `rewardInfo.balance` value if reward tokens have been withdrawn from the contract.

Update

Client's response

Fixed in commit [`e16201cb5f594431c122b3c92c9206523c515abe`]

W-04	Token operations can be front-run in `RecapitalizationManager`
Severity	WARNING
Status	• FIXED

Location

File	Location	Line
RecapitalizationManager.sol	contract `RecapitalizationManager` > function `recapitalizePoolByRewards`	108
RecapitalizationManager.sol	contract `RecapitalizationManager` > function `recapitalizePoolByStackedZun`	134
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `withdrawToken`	86

Description

The functions `recapitalizePoolByRewards`, `recapitalizePoolByStackedZun`, and `withdrawToken` can be front-run to extract profit or prevent losses.

Calling the `recapitalizePoolByRewards` function can be preempted by calling the `distributeRewards` function. This causes the reward tokens to be transferred to the `stakingRewardDistributor` contract.

```
function distributeRewards() external {
    // ...
    stakingRewardDistributor.distribute(address(token_), transferAmount);
    // ...
}
```

By the time the `recapitalizePoolByRewards` function is executed, the contract balance does not contain any reward tokens, which can lead to undesirable results or incorrect operation.

Example scenario:

- The contract accumulates reward tokens over time.
- The malicious actor monitors the contract and anticipates when the `recapitalizePoolByRewards` function might be called.

- Just before the expected call to `recapitalizePoolByRewards`, the malicious actor initiates a transaction calling `distributeRewards`.
- All accumulated reward tokens are transferred to the `stakingRewardDistributor` contract, leaving none in the main pool contract.
- The intended call to `recapitalizePoolByRewards` happens after the malicious actor's transaction.
- Due to the front-running, the contract balance for reward tokens is now empty. `recapitalizePoolByRewards` attempts to sell reward tokens but fails because there are none.

The attacker can exploit a similar logic with `recapitalizePoolByStackedZun` or `withdrawToken`.

The front-running of the `recapitalizePoolByStackedZun` or `withdrawToken` call allows a malicious actor to avoid a loss of funds, which manifests itself in the inability to fully withdraw staked `ZUN` tokens after some of them have been used to recapitalize the pool.

```
function recapitalizePoolByStackedZun(
    uint256 zunAmount,
    IRewardManager rewardManager,
    IPool pool,
    uint256 sid,
    uint256 tid
) external onlyRole(EMERGENCY_ADMIN_ROLE) {

    //...

    stakingRewardDistributor.withdrawToken(zunAmount);

    //...

}
```

Recommendation

We recommend considering the possibility of a front-run and implementing logic to make such attack unprofitable. Using private pools to send such transaction can help in preventing front-running.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

W-05	Mistake in pool token order leads to the incorrect calculations in `StaticCurveLPOracle`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > function `getUSDPrice`	60

Description

The `getUSDPrice` function in the `StaticCurveLPOracle` contract uses the token ID to determine its balance in the pool.

```
uint256 balance = _getBalance(pool, i);
```

Incorrect order of pool tokens in the constructor will result in mismatched token balances, which in turn will result in incorrect calculation of the `getUSDPrice` function.

Recommendation

We recommend taking pool tokens from `curveRegistryCache.coins(pool)` external call instead of the parameter in the `StaticCurveLPOracle` contract constructor.

W-06	Post-recapitalization deposits in `ZUNStakingRewardDistributor` are subject to withdrawal penalty
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `_deposit`	141

Description

The `_deposit` function in the `ZUNStakingRewardDistributor` contract accounts the staked ZUN tokens in the `recapitalizedAmount` variable. When a user attempts to withdraw tokens, they will encounter a penalty proportional to the recapitalization coefficient. The penalty will be lifted once the recapitalized `ZUN` tokens are recovered.

```
function _deposit(uint256 _amount, address _receiver) internal {
    //...
    recapitalizedAmount += _amount - amountReduced;
    //...
}
```

This mechanism is not obvious to the user making a deposit. Common logic suggests that a deposit made after recapitalization should not be subject to a penalty. The user should be able to withdraw the amount of the deposit, unless there has been another recapitalization.

Recommendation

We recommend implementing logic that exempts post-recapitalization deposits from the recapitalization penalty, thereby enabling full withdrawal of these funds.

W-07	Missing parameter validation
Severity	WARNING
Status	• FIXED

Location

File	Location	Line
CrvUsdOracle.sol	contract `CrvUsdOracle` > `constructor` > `genericOracle_`	27
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > `constructor` > `genericOracle_`	28
ZunUsdOracle.sol	contract `ZunUsdOracle` > `constructor` > `genericOracle_`	23

Description

Parameter validation is not performed in the specified constructors. The lack of validation can lead to unpredictable behavior or the occurrence of panic errors.

Recommendation

We recommend implementing validation for parameters to ensure stable and predictable behavior.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

W-08	State update after external call in `ZunamiPool`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZunamiPool.sol	contract `ZunamiPool` > function `withdraw`	276

Description

In the `withdraw` function of the `ZunamiPool` contract, the contract state is updated after the `strategy.withdraw` method is called and the funds have already been transferred to the recipient.

```
function withdraw(
    uint256 sid,
    uint256 stableAmount,
    uint256[POOL_ASSETS] memory tokenAmounts,
    address receiver
) external whenNotPaused onlyRole(CONTROLLER_ROLE) {
    //...
    if (
        !strategy.withdraw(
            receiver == address(0) ? controllerAddr : receiver,
            calcRatioSafe(stableAmount, _strategyInfo[sid].minted),
            tokenAmounts
        )
    ) revert WrongWithdrawParams(sid);

    //...
}
```

Recommendation

We recommend using the Checks-Effects-Interactions pattern to mitigate potential reentrancy attacks.

W-09	Protocol operations may be blocked if conversion declined by slippage
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
FraxEthNativeConverter.sol	contract `FraxEthNativeConverter` > function `applySlippage`	74
SellingCurveRewardManager.sol	contract `SellingCurveRewardManager` > function `checkSlippage`	143
SellingCurveRewardManagerFrxEth.sol	contract `SellingCurveRewardManagerFrxEth` > function `checkSlippage`	137
StableConverter.sol	contract `StableConverter` > function `applySlippage`	101

Description

The mentioned places use the constant `DEFAULT_SLIPPAGE` for conversions. Although the `handle` method accepts a slippage parameter, the value `0` is passed as slippage throughout the codebase, which leads to the use of the constant `DEFAULT_SLIPPAGE`.

In volatile market conditions, this can lead to protocol operations being blocked due to conversion failures if the actual price deviation exceeds the default value even by a small amount.

Recommendation

We recommend adding the possibility to adjust slippage in case it is required by the market conditions.

W-10	Withdraw possible within same block as deposit in `ZunamiPoolControllerBase`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `withdraw`	102

Description

The `withdraw` function in the `ZunamiPoolControllerBase` contract does not check if the `withdrawal` call occurs in the same block as the `deposit` call. This creates an opportunity for an attack using a flashloan, which requires the borrowed funds to be returned in the same transaction.

An attacker can deposit a significant amount of funds into the protocol, perform a series of complex interactions with external pools, and then withdraw them with the possibility of making a large profit at the expense of the protocol.

Recommendation

We recommend eliminating the possibility of withdrawal within the same block as a deposit, for example, by keeping the deposit block number value with each deposit and verifying that the withdraw method is called no earlier than in the following block, similar to the `LockInfo` struct in `ZUNStakingRewardDistributor`. This will prevent malicious actors from using flash loans in potential attacks, as well as various manipulations related to deposits and withdrawals within a single transaction or a single block.

W-11	Unused return value in `ZunamiDepositZap`
Severity	WARNING
Status	• FIXED

Location

File	Location	Line
ZunamiDepositZap.sol	contract `ZunamiDepositZap` > function `deposit`	45

Description

The `deposit` function in the `ZunamiDepositZap` contract does not handle the return value of the `omnipoolController.deposit` call.

```
function deposit(
    uint256[POOL_ASSETS] memory amounts,
    address receiver
) external returns (uint256 shares) {
    if (receiver == address(0)) {
        receiver = msg.sender;
    }

    //...

    omnipoolController.deposit(amounts, address(this));

    uint256 zunStableAmount = IERC20(address(omnipool)).balanceOf(address(this));

    IERC20(address(omnipool)).safeIncreaseAllowance(address(apsController), zunStableAmount);
    return apsController.deposit([zunStableAmount, 0, 0, 0, 0], receiver);
}
```

It should be noted that the current `zunUSD` balance of the `ZunamiDepositZap` contract is used for deposit into the `apsController`. However, this balance may contain tokens other than those that were deposited by the user in the current transaction.

Recommendation

We recommend using the return value of the `deposit` call instead of the current `zunUSD` balance.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

W-12	Unclear rewards receiver in `ZUNStakingRewardDistributor`
Severity	WARNING
Status	• FIXED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `withdraw`	171

Description

In the `withdraw` function of the `ZUNStakingRewardDistributor` contract, the reward recipient address is set to `address(0)`. This means that the reward will always be transferred to the `msg.sender`.

```
function withdraw(
    // ...
    address _tokenReceiver
) external nonReentrant {
    // ...
    _checkpointRewards(msg.sender, totalSupply(), _claimRewards, address(0));
    //...
}
```

The function includes a `_tokenReceiver` parameter that allows users to specify a recipient address for the withdrawn tokens. However, there is no such logic for the reward recipient address.

Recommendation

We recommend implementing logic that allows users to customize the rewards receiver address.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

W-13	Rewards distribution can be manipulated in `ZunamiPoolCompoundController`
Severity	WARNING
Status	• ACKNOWLEDGED

Location

File	Location	Line
[ZunamiPoolCompoundController.sol]()	contract `ZunamiPoolCompoundController` > function `autoCompoundAll`	102

Description

In the function `autoCompoundAll` of the `ZunamiPoolCompoundController` contract, reward tokens are sold for `zunUSD`, and the received `zunUSD` is reinvested in strategies again.

After this action, the share withdraw rate of `ZunamiPoolCompoundController` increases proportionally. Additionally, there are no restrictions for users to deposit and withdraw in the same block.

Possible scenario (with numbers from a test case):

- There are `100` `CVX` reward tokens which have not been compounded yet.
- There is only one holder of `ZunamiPoolCompoundController` shares - it has `200` shares.
- Bob takes a large amount of `zunUSD` using a flash loan - `10000` `zunUSD` in the test case.
- Bob deposits `10000` `zunUSD` into `ZunamiPoolCompoundController` and receives `10000` shares.
- Bob calls the `autoCompoundAll` public method of the `ZunamiPoolCompoundController` contract.
- Bob withdraws his `10000` shares and receives `10463` `zunUSD`.
- Bob returns `10010` `zunUSD` to the flash loan provider (with an interest fee, which is `0.1%`).
- Bob makes a `453` `zunUSD` profit without any risk of failure (if he uses flashbots for transaction execution).

```
it.only('M-03 test: should deposit, withdraw and compound all with flash loan', async () => {
    const {
        admin,
        bob,
        zunamiPool,
        zunamiPoolController,
        zunamiPoolAps,
        zunamiPoolControllerAps,
        strategiesAps,
        stableConverterAps,
    } = await loadFixture(deployFixture);

    await zunamiPoolController.connect(admin).deposit(
        getMinAmountZunUSD('10000'),
        admin.getAddress()
    );

    console.log(
        'zunUSD admin balance:',
        await zunamiPool.balanceOf(admin.address)
    );

    await zunamiPool.transfer(
        bob.address,
        ethers.utils.parseUnits('10000', 'ether')
    );

    console.log(
        'zunUSD bob balance before:',
        await zunamiPool.balanceOf(bob.address)
    );

    // DEPOSIT ------------------------------
    // --------------------------------------

    // admin deposit 200 zunUSD
    for (let i = 0; i < strategiesAps.length; i++) {
        const strategy = strategiesAps[i];
        await zunamiPoolAps.addStrategy(strategy.address);

        await zunamiPoolControllerAps.setDefaultDepositSid(i);

        const zStableBalance = parseUnits('100', 'ether');

        await zunamiPool.approve(zunamiPoolControllerAps.address, zStableBalance);

        await zunamiPoolControllerAps.connect(admin)
            .deposit([zStableBalance, 0, 0, 0, 0], admin.getAddress());
    }
```

```
console.log(
    'apsShares admin balance:',
    await zunamiPoolControllerAps.balanceOf(admin.address)
);

// bob deposit 10000 zunUSD
await zunamiPoolControllerAps.setDefaultDepositSid(0);
const zStableBalance = parseUnits('10000', 'ether');

await zunamiPool.connect(bob).approve(zunamiPoolControllerAps.address, zStableBalance);

await zunamiPoolControllerAps.connect(bob)
    .deposit([zStableBalance, 0, 0, 0, 0], bob.getAddress());

console.log(
    'apsShares bob balance:',
    await zunamiPoolControllerAps.balanceOf(bob.address)
);

// autoCompoundAll ---------------------
// -------------------------------------

await mintTokenTo(
    zunamiPoolControllerAps.address, // zEthFrxEthCurveConvex
    admin,
    '0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B', // CVX
    '0x28C6c06298d514Db089934071355E5743bf21d60', // CVX Vault
    parseUnits('100', 'ether')
);

await zunamiPool.transfer(
    stableConverterAps.address,
    ethers.utils.parseUnits('10000', 'ether')
);

await zunamiPoolControllerAps.autoCompoundAll();

// WITHDRAW zunUSD ---------------------
// -------------------------------------

const withdrawAmount = ethers.utils.parseUnits('10000', 'ether');

await zunamiPoolControllerAps.setDefaultWithdrawSid(0);
```

```
    await zunamiPoolControllerAps.connect(bob).withdraw(
        withdrawAmount,
        [0, 0, 0, 0, 0],
        bob.getAddress()
    );

    console.log(
        'zunUSD bob balance after:',
        await zunamiPool.balanceOf(bob.address)
    );
});
```

Recommendation

We recommend implementing a rewards distribution mechanism that prevents the possibility of taking advantage of the protocol by using large deposits.

2.4 INFO

I-01	Cache array length outside of loop
Severity	INFO
Status	• FIXED

Location

File	Location	Line
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > function `isTokenSupported`	40
ZunDistributor.sol	contract `ZunDistributor` > function `addGauge`	241
RecapitalizationManager.sol	contract `RecapitalizationManager` > function `distributeRewards`	95
ZunamiPool.sol	contract `ZunamiPool` > function `claimRewards`	113
ZunamiPool.sol	contract `ZunamiPool` > function `_mintExtraGains`	129
ZunamiPool.sol	contract `ZunamiPool` > function `addStrategy`	299
ZunamiPool.sol	contract `ZunamiPool` > function `moveFundsBatch`	340

Description

In the mentioned places, the array length is not stored in a variable before the `for` loop is executed. If the length is not cached, the Solidity compiler will read the array length every time during each iteration.

Therefore, if it is a `storage` array, this leads to an additional `sload` operation, and if it is a `memory` array, then an additional `mload` operation.

Recommendation

We recommend caching the array length in the additional variable.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-02	Variables initialized with default value
Severity	INFO
Status	• FIXED

Location

File	Location	Line
RewardTokenManager.sol	contract `RewardTokenManager` > function `_setRewardTokens`	23
RewardTokenManager.sol	contract `RewardTokenManager` > function `_sellRewards`	41
RewardTokenManager.sol	contract `RewardTokenManager` > function `_sellRewards`	54
RewardTokenManager.sol	contract `RewardTokenManager` > function `_sellRewards`	75
CurveNStratBase.sol	contract `CurveNStratBase` > function `checkDepositSuccessful`	39
CurveStratBase.sol	contract `CurveStratBase` > function `checkDepositSuccessful`	39
ERC4626StratBase.sol	contract `ERC4626StratBase` > function `checkDepositSuccessful`	40
VaultStrat.sol	contract `VaultStrat` > function `deposit`	34
VaultStrat.sol	contract `VaultStrat` > function `totalHoldings`	62
VaultStrat.sol	contract `VaultStrat` > function `totalHoldings`	63
VaultStrat.sol	contract `VaultStrat` > function `calcTokenAmount`	77
VaultStrat.sol	contract `VaultStrat` > function `calcTokenAmount`	78
VaultStrat.sol	contract `VaultStrat` > function `transferPortionTokensTo`	88
ZunamiStratBase.sol	contract `ZunamiStratBase` > `constructor`	39
ZunamiStratBase.sol	contract `ZunamiStratBase` > function `withdraw`	125
ZunamiStratBase.sol	contract `ZunamiStratBase` > function `transferTokensOut`	169
ZunamiStratBase.sol	contract `ZunamiStratBase` > function `convertTokensToDynamic`	183
ZunamiStratBase.sol	contract `ZunamiStratBase` > function `fillArrayN`	193
ZunDistributor.sol	contract `ZunDistributor` > function `addGauge`	241
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointRewards`	140

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointReward`	162
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_updateDistribution`	191
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `getPendingReward`	250
RecapitalizationManager.sol	contract `RecapitalizationManager` > function `distributeRewards`	95
ZunamiDepositZap.sol	contract `ZunamiDepositZap` > function `deposit`	37
RewardViewer.sol	contract `RewardViewer` > function `getStakeDaoVaultEarned`	128
ZunamiPool.sol	contract `ZunamiPool` > function `_setTokens`	77
ZunamiPool.sol	contract `ZunamiPool` > function `claimRewards`	113
ZunamiPool.sol	contract `ZunamiPool` > function `totalHoldings`	169
ZunamiPool.sol	contract `ZunamiPool` > function `doDepositStrategy`	217
ZunamiPool.sol	contract `ZunamiPool` > function `processSuccessfulDeposit`	236
ZunamiPool.sol	contract `ZunamiPool` > function `moveFundsBatch`	347
ZunamiPoolCompoundController.sol	contract `ZunamiPoolCompoundController` > function `depositPool`	147
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `deposit`	80

Description

In the mentioned places, there is redundant initialization of variables with default values.

Recommendation

We recommend removing redundant initialization.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-03	Parameter `_totalSupply` is redundant
Severity	INFO
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointRewards`	123
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointRewards`	141
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointReward`	156
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointReward`	159
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_updateDistribution`	187
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `distribute`	217
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `claim`	222
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_update`	302
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_update`	303
StakingRewardDistributor.sol	contract `StakingRewardDistributor` > function `deposit`	23
StakingRewardDistributor.sol	contract `StakingRewardDistributor` > function `withdraw`	45
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `_deposit`	134
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `withdraw`	171

Description

In the mentioned locations the parameter `_totalSupply` is passed down the call stack until the `_updateDistribution` function, where it is used.

```
function _updateDistribution(
    uint256 _tid,
    uint256 _totalSupply
) internal returns (uint256 distribution) {
    // ...
}
```

However, the only value passed with this parameter is `totalSupply()`, which can be obtained directly in the function `_updateDistribution`.

Recommendation

We recommend removing redundant parameter to keep the codebase clean.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-04	Magic numbers in `BaseStakingRewardDistributor`, `CrvUsdApsConvexCurveStratBase`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_checkpointReward`	166
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_updateDistribution`	197
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `getPendingReward`	255
CrvUsdApsConvexCurveStratBase.sol	contract `CrvUsdApsConvexCurveStratBase` > function `convertCurvePoolTokenAmounts`	75
CrvUsdApsConvexCurveStratBase.sol	contract `CrvUsdApsConvexCurveStratBase` > function `convertAndApproveTokens`	84
CrvUsdApsConvexCurveStratBase.sol	contract `CrvUsdApsConvexCurveStratBase` > function `_inflate`	125
CrvUsdApsConvexCurveStratBase.sol	contract `CrvUsdApsConvexCurveStratBase` > function `_deflate`	166

Description

In the mentioned locations literal values with unexplained meaning are used to perform calculations.

Recommendation

We recommend defining a constant for every magic number, giving it a clear and self-explanatory name.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-05	Token address validation is redundant in `BaseStakingRewardDistributor`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `isRewardTokenAdded`	246

Description

In the function `isRewardTokenAdded` of the contract `BaseStakingRewardDistributor` token address validation is redundant as the address is never updated after reward token is added to the storage.

```
function isRewardTokenAdded(address _token) public view returns (bool) {
    uint256 tid = rewardTokenTidByAddress[_token];
    return rewardTokenInfo.length > tid && address(rewardTokenInfo[tid].token) == _token;
}
```

Recommendation

We recommend removing redundant validation.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-06	Redundant cast in `ZUNStakingRewardDistributor`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `_deposit`	136

Description

In the function `_deposit` of the contract `ZUNStakingRewardDistributor` the cast `address(msg.sender)` is redundant.

```
token.safeTransferFrom(address(msg.sender), address(this), _amount);
```

Recommendation

We recommend changing the statement to `msg.sender` to keep the codebase clean and save gas.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-07	Unused function in `ZunUsdOracle`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZunUsdOracle.sol	contract `ZunUsdOracle` > function `_median`	46

Description

The function `_median` of contract `ZunUsdOracle` is not used in the contract codebase.

```
function _median(uint256 a, uint256 b, uint256 c) internal pure returns (uint256) {
    if ((a >= b && a <= c) || (a >= c && a <= b)) return a;
    if ((b >= a && b <= c) || (b >= c && b <= a)) return b;
    return c;
}
```

Recommendation

We recommend removing redundant function.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-08	Typo in `ZUNStakingRewardDistributor`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor`	26

Description

In the `ZUNStakingRewardDistributor` contract, there is a spelling error: the word `RATION` is used instead of `RATIO`.

```
uint256 public constant RATION_DENOMINATOR = 1e18;
```

Recommendation

We recommend fixing the typo by replacing instances of the word `RATION` with `RATIO`.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-09	Use error type in `CrvUsdOracle`, `StaticCurveLPOracle`, `ZunUsdOracle`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
CrvUsdOracle.sol	contract `CrvUsdOracle` > function `getUSDPrice`	32
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > `constructor`	34
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > function `getUSDPrice`	59
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > function `getUSDPrice`	61
StaticCurveLPOracle.sol	contract `StaticCurveLPOracle` > function `setImbalanceThreshold`	82
ZunUsdOracle.sol	contract `ZunUsdOracle` > function `getUSDPrice`	28

Description

In the mentioned locations, `require` is used for error handling.

Recommendation

We recommend using the error type instead of `require` to maintain the same code style throughout the entire project.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-10	Unused import in `StaticCurveLPOracle`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
StaticCurveLPOracle.sol	None	10

Description

The import in the mentioned contract is redundant and can be removed.

```
import '../../interfaces/vendor/ICurvePoolV0.sol';
```

Recommendation

We recommend removing unused import to keep the codebase clean.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-11	Inconsistent interface location in `CrvUsdOracle`, `ZunUsdOracle`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
CrvUsdOracle.sol	interface `ICurvePriceOracle` > function `price_oracle`	7
ZunUsdOracle.sol	interface `ICurvePriceOracleNG` > function `price_oracle`	7

Description

In the mentioned locations, interfaces are declared in contract implementation files. However, there is a separate `interfaces` folder for interfaces in the project.

Recommendation

We recommend moving the interfaces to the `interfaces` folder.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-12	Single `userLocks` withdraw in `ZUNStakingRewardDistributor`
Severity	INFO
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZUNStakingRewardDistributor.sol	contract `ZUNStakingRewardDistributor` > function `withdraw`	155

Description

The `withdraw` function in the `ZUNStakingRewardDistributor` contract currently only allows withdrawing tokens from a single userLock in a single transaction.

```
function withdraw(
    uint256 _lockIndex,
    bool _claimRewards,
    address _tokenReceiver
) external nonReentrant {
    LockInfo[] storage locks = userLocks[msg.sender];
    if (locks.length <= _lockIndex) revert LockDoesNotExist();

    LockInfo storage lock = locks[_lockIndex];

    // ...

    lock.untilBlock = 0;

    // ...

    emit Withdrawn(msg.sender, _lockIndex, amount, amountReduced, transferredAmount);
}
```

This functionality can negatively affect the user experience.

Recommendation

We recommend implementing functionality to allow withdrawing tokens from several `userLocks` in one transaction.

I-13	No access control in `ApproveGauge`
Severity	INFO
Status	• ACKNOWLEDGED

Location

File	Location	Line
ApproveGauge.sol	contract `ApproveGauge` > function `distribute`	22

Description

The `distribute` function in the `ApproveGauge` contract does not implement access control, which allows any user to call it to increase the allowance for any amount.

```
function distribute(uint256 amount) external virtual {
    TOKEN.safeIncreaseAllowance(RECEIVER, amount);
}
```

Recommendation

We recommend adding access control to the function.

I-14	Unused import in `BaseStakingRewardDistributor`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	None	8

Description

In the contract `BaseStakingRewardDistributor`, the import of `ERC20VotesUpgradeable` is redundant as it is not used anywhere in the contract. The import is used in the derived contract `ZUNStakingRewardDistributor`.

Recommendation

We recommend moving the import statement to the `ZUNStakingRewardDistributor.sol` file.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-15	Lack of access contol, posssible sandwich attack in `ZunamiPoolCompoundController`
Severity	INFO
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZunamiPoolCompoundController.sol	contract `ZunamiPoolCompoundController` > function `autoCompoundAll`	102

Description

The `autoCompoundAll` function in the `ZunamiPoolCompoundController` contract lacks access control, which allows any user to call it. This function performs the selling of rewards and the depositing of the received funds into the pool.

```
function autoCompoundAll() public whenNotPaused nonReentrant {
    claimPoolRewards(address(this));

    IERC20 feeToken = pool.token(feeTokenId);
    if (address(feeToken) == address(0)) revert ZeroFeeTokenAddress();

    uint256 received = sellRewards(feeToken);
    if (received == 0) return;

    uint256[POOL_ASSETS] memory amounts;
    amounts[feeTokenId] = received;
    feeToken.safeTransfer(address(pool), amounts[feeTokenId]);

    uint256 depositedValue = pool.deposit(defaultDepositSid, amounts, address(this));

    emit AutoCompoundedAll(depositedValue);
}
```

An attacker can exploit this vulnerability to manipulate pools with conversions using a sandwich attack during the `AutoCompoundAll` function call.

Recommendation

We recommend adding the access control and sending the transaction through the private pool to avoid the attack.

I-16	Users not able to manage position when protocol paused in `ZunamiPoolControllerBase`
Severity	INFO
Status	• ACKNOWLEDGED

Location

File	Location	Line
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `withdraw`	102

Description

In the function `withdraw` of the contract `ZunamiPoolControllerBase`, the modifier `whenNotPaused` prevents the users from withdrawing funds when the contract is paused.

```
function withdraw(
    uint256 shares,
    uint256[POOL_ASSETS] memory minTokenAmounts,
    address receiver
) external whenNotPaused nonReentrant {
    if (receiver == address(0)) {
        receiver = _msgSender();
    }
    withdrawPool(_msgSender(), shares, minTokenAmounts, receiver);
}
```

A reasonable expectation from users is the ability to withdraw their funds at any moment, unless the protocol workflow requires accounting for socialized losses that may be delayed, which is not the case according to the business logic of the protocol.

Recommendation

We recommend allowing users to withdraw funds when the protocol is paused.

I-17	High gas consumption of `transfer` function in `BaseStakingRewardDistributor`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `_update`	302

Description

In the `_update` function of the `BaseStakingRewardDistributor` contract, a significant amount of additional logic is executed with each `vlZUN` token transfer, leading to substantial gas consumption for the `transfer` function. In some scenarios, it reaches `239k` gas, which is considerable for usage on the Ethereum mainnet.

```
  StakingRewardDistributor tests
Gas:  BigNumber { value: "110280" }
Gas:  BigNumber { value: "93180" }
    :heavy_check_mark: update staking balance by moving staking token (378ms)
Gas:  BigNumber { value: "239522" }
Gas:  BigNumber { value: "115080" }
    :heavy_check_mark: should distribute reward token if user send some LP to another
(10629ms)
```

Recommendation

We recommend refactoring the reward distribution logic to make transfers of `vlZUN` cheaper.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-18	Redundant use of `_msgSender()`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZunamiPoolAccessControl.sol	contract `ZunamiPoolAccessControl` > modifier `onlyZunamiPool`	18
ZunamiStratBase.sol	contract `ZunamiStratBase` > function `withdrawAll`	159
BaseStakingRewardDistributor.sol	contract `BaseStakingRewardDistributor` > function `withdrawStuckToken`	276
RecapitalizationManager.sol	contract `RecapitalizationManager` > function `withdrawStuckToken`	195
ZunamiPool.sol	contract `ZunamiPool` > function `deposit`	192
ZunamiPool.sol	contract `ZunamiPool` > function `withdraw`	262
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `deposit`	76
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `deposit`	83
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `withdraw`	108
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `withdraw`	110

Description

In the mentioned locations, the call to `_msgSender()` is used instead of `msg.sender`. This call is redundant as the contract does not implement the required setup for using the Gas Station Network.

Recommendation

We recommend replacing the call `_msgSender()` with `msg.sender` to make the code more readable and save gas.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-19	`++i` is more efficient than `i++`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZunDistributor.sol	contract `ZunDistributor` > function `addGauge`	241
ZunDistributor.sol	contract `ZunDistributor` > function `deleteGauge`	257

Description

In the mentioned locations and throughout the entire protocol codebase, `i++` is used for loop iteration. However, using `++i` is more gas-efficient than `i++`, especially in loops.

Recommendation

We recommend using `++i` instead of `i++` to reduce gas consumption.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

I-20	Constant usage is cheaper in `ZunamiDepositZap`, `ZunamiPool`, `ZunamiPoolControllerBase`
Severity	INFO
Status	• FIXED

Location

File	Location	Line
ZunamiDepositZap.sol	contract `ZunamiDepositZap` > function `deposit`	37
ZunamiPool.sol	contract `ZunamiPool` > function `doDepositStrategy`	217
ZunamiPoolControllerBase.sol	contract `ZunamiPoolControllerBase` > function `deposit`	80

Description

In the mentioned locations, `amounts.length` is used in loop iteration. However, the length of the `amounts` array is constant and always equals `POOL_ASSETS`.

```
function deposit(
    uint256[POOL_ASSETS] memory amounts,
    // ...
) external returns (uint256 shares) {
    // ...
    for (uint256 i = 0; i < amounts.length; i++) {
    //...
}
```

Recommendation

We recommend using the `POOL_ASSETS` constant instead of `amounts.length` for loop iterations to reduce gas consumption.

Update

Client's response

Fixed in commit `e16201cb5f594431c122b3c92c9206523c515abe`

3 CONCLUSION

OXORIO

The Zunami protocol security audit identified a range of vulnerabilities classified as critical, major, warning, and informational.

The accompanying report details specific remediation measures for each vulnerability. Addressing these findings will ensure the protocol's security and stability for its user base.

The following table contains all the findings identified during the audit, grouped by statuses and severity levels:

Severity	FIXED	ACKNOWLEDGED	Total
CRITICAL	1	0	1
MAJOR	2	0	2
WARNING	5	8	13
INFO	16	4	20
Total	24	12	36

THANK YOU FOR CHOOSING

OXORIO