

# Zunami Protocol

smart contracts  
preliminary audit report  
for internal use only

---

March 2023



[hashex.org](https://hashex.org)



[contact@hashex.org](mailto:contact@hashex.org)

# Contents

|   |    |
|---|----|
| 1. Disclaimer                               | 3  |
| 2. Overview                                 | 4  |
| 3. Found issues                             | 7  |
| 4. Contracts                                | 11 |
| 5. Conclusion                               | 30 |
| Appendix A. Issues' severity classification | 31 |
| Appendix B. List of examined issue types    | 32 |

# 1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below - please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author ([hashex.org](https://hashex.org)).

## 2. Overview

HashEx was commissioned by the Zunami Protocol team to perform an audit of their smart contract. The audit was conducted between 26/02/2023 and 03/03/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @ZunamiProtocol/ZunamiProtocol GitHub repo after the [894b52d](#) commit.

Only active **stakingFraxbp** and **stakedao** strategies, **rewardManager**, and **utility** folders were audited in the contracts/strategies folder. The folder **crosschain** was also excluded from this audit's scope.

**Update.** Recheck was done after the commit [0e92770](#).

### 2.1 Summary

|              |   |
|--------------|---|
| Project name | Zunami Protocol   |
| URL          | <a href="https://www.zunami.io/">https://www.zunami.io/</a> |
| Platform     | Ethereum  |
| Language     | Solidity  |

## 2.2 Contracts

| Name                            | Address   |
|---------------------------------|---|
| StakingFraxCurveConvexStratBase | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/curve/convex/stakingFraxbp/StakingFraxCurveConvexStratBase.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/curve/convex/stakingFraxbp/StakingFraxCurveConvexStratBase.sol</a> |
| XAIStakingFraxCurveConvex       | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/curve/convex/stakingFraxbp/XAIStakingFraxCurveConvex.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/curve/convex/stakingFraxbp/XAIStakingFraxCurveConvex.sol</a>             |
| CurveStakeDaoStratBase          | 0xE9ACC52411710e7fAa99c6d41C7CFE07AFfbd2CB  |
| CurveStakeDaoExtraStratBase     | 0xE9ACC52411710e7fAa99c6d41C7CFE07AFfbd2CB  |
| CurveStakeDaoStrat2             | 0xE9ACC52411710e7fAa99c6d41C7CFE07AFfbd2CB  |
| MIMCurveStakeDao                | 0xE9ACC52411710e7fAa99c6d41C7CFE07AFfbd2CB  |
| SellingUniswapRewardManager     | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/rewardManager/SellingUniswapRewardManager.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/rewardManager/SellingUniswapRewardManager.sol</a>                                   |
| SellingCurveRewardManager       | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/rewardManager/SellingCurveRewardManager.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/rewardManager/SellingCurveRewardManager.sol</a>                                       |
| RebalancingStrat                | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/utility/RebalancingStrat.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/utility/RebalancingStrat.sol</a>   |
| StableConverter                 | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/utility/StableConverter.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/strategies/utility/StableConverter.sol</a>   |
| Constants                       | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/utills/Constants.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/utills/Constants.sol</a>   |

---

|              |   |
|--------------|---|
| ConstantsBSC | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/Utils/ConstantsBSC.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/Utils/ConstantsBSC.sol</a> |
| Zunami       | <a href="https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/Zunami.sol">https://github.com/ZunamiProtocol/ZunamiProtocol/blob/0e927705ec9492ff02be71964df9e1b73a00fae5/contracts/Zunami.sol</a>                         |

---

### 3. Found issues



|        |          |
|--------|----------|
| High   | 6 (17%)  |
| Medium | 4 (11%)  |
| Low    | 15 (42%) |
| Info   | 11 (30%) |

#### C1. StakingFraxCurveConvexStratBase

| ID    | Severity | Title   | Status            |
|-------|----------|---|-------------------|
| C1-01 | High     | Exaggerated owner rights                                | ☑ Acknowledged    |
| C1-02 | Medium   | Deposit is open for public                              | ☑ Acknowledged    |
| C1-03 | Low      | Swaps with 100% slippage                                | ☑ Acknowledged    |
| C1-04 | Low      | Token transfer results are not checked in sellRewards() | ✅ Resolved        |
| C1-05 | Low      | Gas optimization  | ✅ Resolved        |
| C1-06 | Info     | On-chain slippage check                                 | ☑ Acknowledged    |
| C1-07 | Info     | decimalMultipliers should replace constants             | ☑ Acknowledged    |
| C1-08 | Info     | State variable default visibility                       | ✅ Resolved        |
| C1-09 | Info     | Lack of events  | 🔧 Partially fixed |

### C3. CurveStakeDaoStratBase

| ID    | Severity | Title                      | Status            |
|-------|----------|----------------------------|-------------------|
| C3-01 | High     | Exaggerated owner rights   | ☑ Acknowledged    |
| C3-02 | Medium   | Deposit is open for public | ☑ Acknowledged    |
| C3-03 | Low      | Gas optimization           | 🔧 Partially fixed |
| C3-04 | Info     | Spelling error             | ✅ Resolved        |
| C3-05 | Info     | Lack of events             | 🔧 Partially fixed |

### C4. CurveStakeDaoExtraStratBase

| ID    | Severity | Title  | Status     |
|-------|----------|--|------------|
| C4-01 | Low      | Token transfer results are not checked in sellRewardsExtra() | ✅ Resolved |
| C4-02 | Low      | Gas optimization   | ✅ Resolved |
| C4-03 | Info     | Spelling error   | ✅ Resolved |

### C5. CurveStakeDaoStrat2

| ID    | Severity | Title                         | Status         |
|-------|----------|-------------------------------|----------------|
| C5-01 | Low      | On-chain slippage calculation | ☑ Acknowledged |
| C5-02 | Low      | Gas optimization              | ✅ Resolved     |



## C7. SellingUniswapRewardManager

| ID    | Severity | Title                             | Status         |
|-------|----------|-----------------------------------|----------------|
| C7-01 | ● Low    | Gas optimization                  | ✓ Resolved     |
| C7-02 | ● Low    | Misuse of deadline swap parameter | ✓ Acknowledged |

## C8. SellingCurveRewardManager

| ID    | Severity | Title            | Status     |
|-------|----------|------------------|------------|
| C8-01 | ● Low    | Gas optimization | ✓ Resolved |

## C9. RebalancingStrat

| ID    | Severity | Title                      | Status            |
|-------|----------|----------------------------|-------------------|
| C9-01 | ● High   | Owner exaggerated rights   | ✓ Acknowledged    |
| C9-02 | ● Medium | Deposit is open for public | ✓ Acknowledged    |
| C9-03 | ● Low    | Gas optimization           | ✚ Partially fixed |

## C10. StableConverter

| ID     | Severity | Title            | Status     |
|--------|----------|------------------|------------|
| C10-01 | ● Low    | Gas optimization | ✓ Resolved |

## C13. Zunami

| ID     | Severity | Title  | Status             |
|--------|----------|--|--------------------|
| C13-01 | ● High   | Default deposit/withdraw pid change concerns | ☑ Acknowledged     |
| C13-02 | ● High   | Lacks of validation on strategy duplication  | ☑ Acknowledged     |
| C13-03 | ● High   | Excessive owner's rights                     | ☑ Acknowledged     |
| C13-04 | ● Medium | Fees may reach up to 100%                    | ✔ Resolved         |
| C13-05 | ● Low    | Events problems                              | ⚙️ Partially fixed |
| C13-06 | ● Low    | Gas optimization                             | ⚙️ Partially fixed |
| C13-07 | ● Info   | Supported tokens constraints                 | ☑ Acknowledged     |
| C13-08 | ● Info   | Inconsistent constant variable usage         | ✔ Resolved         |
| C13-09 | ● Info   | Spelling errors                              | ✔ Resolved         |
| C13-10 | ● Info   | Pauser role proposal                         | ☑ Acknowledged     |

## 4. Contracts

### C1. StakingFraxCurveConvexStratBase

#### Overview

The abstract contract implementing logic for Frax-Curve-Convex strategy. Deposited tokens are firstly swapped to USDC, then strategy sends them to Frax-USDC Curve pool, received from previous staking iteration liquidity tokens are sent to crvFrax pool, and lastly, acquired LP tokens are stored in Vault contract.

#### Issues

##### C1-01 Exaggerated owner rights

● High

☑ Acknowledged

The `setZunami()` function allows the owner to update Zunami contract address, which has access to withdrawals. In other words, owner has full access to all strategies funds by setting `zunami` address to EOA or malicious contract.

```
function setZunami(address zunamiAddr) external onlyOwner {
    zunami = IZunami(zunamiAddr);
}
```

The `setRewardManager()` and `setStableConverter()` functions also grant contract's owner access to set malicious siphoning contracts as RewardManager and/or StableConverter.

#### Recommendation

Consider making these functions initializers, i.e. being callable only once.

Other way is to renounce the ownership or transfer it to a Timelock-like contract with MultiSig admin.

## C1-02 Deposit is open for public

● Medium

✔ Acknowledged

The `deposit()` function is open for public calls, but withdraw is possibly only via Zunami contract, meaning any user, who deposits directly, will lose his tokens.

```
function deposit(uint256[3] memory amounts)
    external returns (uint256) {
        ...
    }

function withdraw(
    address withdrawer,
    uint256 userRatioOfCrvLps,
    uint256[3] memory tokenAmounts,
    WithdrawalType withdrawalType,
    uint128 tokenIndex
) external virtual onlyZunami returns (bool) {
    ...
}
```

### Recommendation

Consider adding deposit restrictions or at least document function's behavior in NatSpec description.

## C1-03 Swaps with 100% slippage

● Low

✔ Acknowledged

The `swapTokenToUSDC()` and `swapUSDCToToken()` functions don't receive slippage parameter (in amountMin form) off-chain, making them vulnerable to sandwich attacks.

## C1-04 Token transfer results are not checked in `sellRewards()`

● Low

✔ Resolved

It is recommended to check returned values of token transfer calls or use [SafeERC20](#) library in case a token violates the [ERC20](#) standard and returns false, but not reverts the transaction.

## C1-05 Gas optimization

● Low

✔ Resolved

- a. The variables `cvxPoolPID`, `fraxUsdcPool`, `fraxUsdcPoolLp`, `crvFraxTokenPool`, `crvFraxTokenPoolLp` should be declared as immutables, `lockingIntervalSec` - as constant;
- b. Multiple reads of `_config.tokens[]` and `feeTokenId` in the `transferAllTokensOut()` and `transferZunamiAllTokens()` functions;
- c. Unchecked math could be used in the `calcTokenDecimalsMultiplier()` function;
- d. Multiple reads from storage of `_config.rewards.length`, `_config.rewards[]`, `_config.tokens[feeTokenId]`, `rewardManager` in the `sellRewards()` function;
- e. Double read from storage of `feeTokenId` in the `autoCompound()` function;
- f. Multiple reads from storage of `feeTokenId`, `managementFees`, `_config.tokens[]` in the `claimManagementFees()` function;
- g. Multiple reads from storage of `stakingVault`, `rewardManager`, `_config.tokens[]`, `feeTokenId` in the `totalHoldings()` function;
- h. Multiple reads from storage of `fraxUsdcPool` variable in the `checkDepositSuccessful()` function (not relevant if `fraxUsdcPool` is immutable);
- i. Multiple reads from storage of `_config.tokens[]`, `fraxUsdcPool`, `crvFraxTokenPool` variables in the `depositPool()` function (not fully relevant if `fraxUsdcPool`, `crvFraxTokenPool` are immutables);
- j. Triple read from storage of `stakingVault` variable in the `stakeCurveLp()` function;
- k. Double read from storage of `stakingVault` variable in the `calcWithdrawOneCoin()` function;
- l. Double read from storage of `stakingVault` variable in the `calcCrvLps()` function;
- m. Multiple reads from storage of `_config.tokens[]`, `stableConverter` variables in the `swapUSDCToToken()` function;

n. Multiple reads from storage of `stableConverter` variables in the `swapTokenToUSDC()` function;

### C1-06 On-chain slippage check

● Info

☑ Acknowledged

The `checkDepositSuccessful()` function checks the min amount during deposit. It doesn't receive any off-chain data for this check, making it useless, i.e. either always passing or always failing.

### C1-07 decimalMultipliers should replace constants

● Info

☑ Acknowledged

The `1e12` constant in the `checkDepositSuccessful()` and `convertZunamiTokensToFraxUsdcs()` functions should be replaced with `decimalsMultipliers` variable, which can be made constant in order to save gas.

### C1-08 State variable default visibility

● Info

✅ Resolved

No explicit visibility is specified for the variables `FRAX_USDC_POOL_USDC_ID`, `FRAX_USDC_POOL_USDC_ID_INT`, `CRVFRAX_TOKEN_POOL_CRVFRAX_ID`, `CRVFRAX_TOKEN_POOL_CRVFRAX_ID_INT`, `CRVFRAX_TOKEN_POOL_TOKEN_ID`.

### C1-09 Lack of events

● Info

🔧 Partially fixed

We recommended emitting events on important value changes to be easily tracked off-chain. No events are emitted in `updateMinDepositAmount()`, `setZunami()`, `setFeeTokenId()`, `changeFeeDistributor()`, `lockLonger()`.

## C2. XAISTakingFraxCurveConvex

### Overview

The main contract in Frax-Curve-Convex strategy inheritance scheme. It is derived from StakingFraxCurveConvexStratBase extending it with constructor that receives config for initialization of StakingFraxCurveConvexStratBase inherited part.

No issues were found.

## C3. CurveStakeDaoStratBase

### Overview

The first contract in Curve-StakeDAO strategy inheritance scheme. It declares external functions for interaction with the strategy and contains important governance methods. During the strategy USD stablecoins are staked in a Curve pool then received liquidity tokens are sent to StakeDAO to maximize yeild.

### Issues

#### C3-01 Exaggerated owner rights

● High

✓ Acknowledged

a. If the owner account is compromised, a malefactor can replace **rewardManager** with malicious one using the **setRewardManager()** function. This may allow the malefactor to assign some part or all reward tokens.

```
function setRewardManager(address rewardManagerAddr) external onlyOwner {  
    rewardManager = IRewardManager(rewardManagerAddr);  
}
```

b. If the owner account is compromised, a malefactor can change **zunami** address using **setZunami()** function. This may allow the malefactor to withdraw all strategy's funds with

`withdrawAll()` method.

```
function setZunami(address zunamiAddr) external onlyOwner {
    zunami = IZunami(zunamiAddr);
}
```

## Recommendation

Consider making these functions initializers, i.e. being callable only once.

Other way is to renounce the ownership or transfer it to a Timelock-like contract with MultiSig admin.

### C3-02 Deposit is open for public

● Medium

☑ Acknowledged

The `deposit()` function is open for public calls, but withdraw is possible only via Zunami contract, meaning any user, who deposits directly, will lose his tokens.

```
function deposit(uint256[3] memory amounts) external returns (uint256) {
    ...
}

function withdraw(
    address withdrawer,
    uint256 userRatioOfCrvLps, // multiplied by 1e18
    uint256[3] memory tokenAmounts,
    WithdrawalType withdrawalType,
    uint128 tokenIndex
) external virtual onlyZunami returns (bool) {
    ...
}
```



## Recommendation

Consider adding deposit restrictions or at least document function's behavior in NatSpec description.

### C3-03 Gas optimization

● Low

🔧 Partially fixed

- a. The variables `vault`, `poolLP` should be declared immutables;
- b. Multiple reads of `_config.tokens[]` and `feeTokenId` in the `transferAllTokensOut()` and `transferZunamiAllTokens()` functions;
- c. Unchecked math could be used in the `calcTokenDecimalsMultiplier()` function;
- d. Multiple reads from storage of `_config.rewards.length`, `_config.rewards[]`, `_config.tokens[feeTokenId]`, `rewardManager` in the `sellRewards()` function;
- e. Double read from storage of `feeTokenId` in the `autoCompound()` function;
- f. Multiple reads from storage of `vault`, `rewardManager`, `_config.tokens[]`, `feeTokenId` in the `totalHoldings()` function (not relevant if `vault` is immutable);
- g. Multiple reads from storage of `feeTokenId`, `managementFees`, `_config.tokens[]` in the `claimManagementFees()`.

### C3-04 Spelling error

● Info

✅ Resolved

Typo in `iterfaces`.

### C3-05 Lack of events

● Info

🔧 Partially fixed

We recommended emitting events on important value changes to be easily tracked off-chain. No events are emitted in `updateMinDepositAmount()`, `setZunami()`, `setFeeTokenId()`, and `changeFeeDistributor()`.

## C4. CurveStakeDaoExtraStratBase

### Overview

The second contract in Curve-StakeDAO strategy inheritance scheme derived from CurveStakeDaoStratBase. It implements getter for total USD holdings in the strategy, methods to sell accumulated reward tokens and withdraw all funds to Zunami contract.

### Issues

**C4-01 Token transfer results are not checked in `sellRewardsExtra()`** ● Low ✓ Resolved

---

It is recommended to check returned values of token transfer calls or use [SafeERC20](#) library in case a token violates the [ERC20](#) standard and returns false, but not reverts the transaction.

**C4-02 Gas optimization** ● Low ✓ Resolved

---

- a. The variables `token`, `extraRewardToken` should be declared as immutables;
- b. Multiple reads of `extraRewardToken`, `feeTokenId` in the `totalHoldings()` function (not relevant if `extraRewardToken` is immutable);
- c. Multiple reads of `extraRewardToken` in the `sellRewardsExtra()` function (not relevant if `extraRewardToken` is immutable);

**C4-03 Spelling error** ● Info ✓ Resolved

---

Typos in `strategys`.

## C5. CurveStakeDaoStrat2

### Overview

The third contract in Curve-StakeDAO strategy inheritance scheme derived from CurveStakeDaoExtraStratBase. It defines auxiliary functions used while deposit and withdrawal operations.

### Issues

#### C5-01 On-chain slippage calculation

● Low

✔ Acknowledged

`checkDepositSuccessful()` function is called during deposit to verify USD price of liquidity tokens to be received satisfy slippage requirements. However, slippage check uses on-chain data of liquidity token price and amount of liquidity tokens on output, which calculations are both based on the current pool balances. As the result slippage check will either always fail or pass depending on `minDepositAmount` value and disregarding `amountsTotal`.

#### C5-02 Gas optimization

● Low

✔ Resolved

- The variables `pool3`, `pool3LP`, `pool1` should be declared immutables;
- Multiple reads from storage of `pool3` variable in the `checkDepositSuccessful()` function (not relevant if `pool3` is immutable);
- Multiple reads from storage of `pool3` variable in the `depositPool()` function (not relevant if `pool3` is immutable);
- Double read from storage of `pool1`, `pool3` variables in the `calcCrvLps()` function (not relevant if `pool1`, `pool3` are immutables);
- Double read from storage of `pool3LP` variable in the `removeCrvLps()` function (not relevant if `pool3` is immutable);

f. Double read from storage of pool variable in the `sellToken()` function (not relevant if `pool` is immutable).

## C6. MIMCurveStakeDao

### Overview

The main contract in Curve-StakeDAO strategy inheritance scheme. It is derived from `CurveStakeDaoStrat2` extending it with constructor that receives config for initialization of `CurveStakeDaoStrat2` inherited part.

No issues were found.

## C7. SellingUniswapRewardManager

### Overview

The contract helps to convert tokens using Uniswap-like pools and estimate ultimately received targeted token amount after the swaps.

### Issues

#### C7-01 Gas optimization

● Low

✓ Resolved

a. The variables `router`, `defaultSlippage`, and `middleSwapToken` should be declared as immutables or constants;

b. Multiple reads of `router` in the `handle()` function (not relevant if `router` is declared immutable).

## C7-02 Misuse of deadline swap parameter

● Low

✔ Acknowledged

Deadline parameter of Uniswap-like router can't be calculated on-chain, it must be received in function parameters or external Oracle, e.g. checking current `block.timestamp` against timestamp of Chainlink latest data.

```
function handle(
    address reward,
    uint256 amount,
    address feeToken
) public {
    ...
    uint256[] memory amounts = router.swapExactTokensForTokens(
        amount,
        0,
        fromAddressArr3([reward, middleSwapToken, feeToken]),
        msg.sender,
        block.timestamp + Constants.TRADE_DEADLINE
    );
    ...
}
```

## C8. SellingCurveRewardManager

### Overview

The contract helps to convert reward tokens on one of three supported stablecoins using Curve pools and estimate ultimately received targeted token amount after the swaps.

### Issues

#### C8-01 Gas optimization

● Low

✔ Resolved

a. The variables `tricrypto2`, `defaultSlippage`, `stableConverter`, `zlp`, `uzd`, and `feeCollector` should be declared as immutables or constants;

b. Multiple reads of `tricrypto2` in the `handle()` function (not relevant if `tricrypto2` is declared immutable).

## C9. RebalancingStrat

### Overview

The contract implements basic strategies' behavior. Is not in use in any of the audited contracts.

### Issues

#### C9-01 Owner exaggerated rights

● High✓ Acknowledged

The owner can accidentally withdraw one of strategy held tokens with `withdrawStuckToken()` or `withdrawStuckTokenTo()` method.

#### C9-02 Deposit is open for public

● Medium✓ Acknowledged

The `deposit()` function is open for public calls, but withdraw is possibly only via Zunami contract, meaning any user, who deposits directly, will lose his tokens.

```
function deposit(uint256[3] memory amounts) external returns (uint256) {
    ...
}

function withdraw(
    address withdrawer,
    uint256 userRatioOfCrvLps, // multiplied by 1e18
    uint256[3] memory tokenAmounts,
    WithdrawalType withdrawalType,
    uint128 tokenIndex
) external virtual onlyZunami returns (bool) {
    ...
}
```

```
}
```

## Recommendation

Consider adding deposit restrictions or at least document function's behavior in NatSpec description.

### C9-03 Gas optimization

● Low

🔧 Partially fixed

- a. Unnecessary reads from storage of `tokens[]` in the constructor;
- b. Double reads from storage of `tokens[]` in the `transferAllTokensTo()`, `transferPortionTokensTo()` functions;
- c. Unchecked math could be used in the `calcTokenDecimalsMultiplier()` function.

## C10. StableConverter

### Overview

The contract helps to convert one of three supported stablecoins to another and estimate ultimately received targeted token amount after the swaps.

### Issues

#### C10-01 Gas optimization

● Low

✅ Resolved

- a. The variables `curve3Pool` and `defaultSlippage` should be declared as constants or immutables;
- b. Double read from storage of `curve3Pool` in the `handle()` function (not relevant if `curve3Pool` is declared immutable);

## C11. Constants

### Overview

The library stores constants of Curve and Convex pools addresses leveraged by strategies and their PIDs as well as token tickers and trade deadline.

No issues were found.

## C12. ConstantsBSC

### Overview

The library stores constants of BUSD, USDT, and PancakeSwap router addresses for Binance Smart Chain.

No issues were found.

## C13. Zunami

### Overview

The contract is an [ERC20](#) standard implementation. It provides an entrypoint for users willing to invest in the strategies. After deposit, user receives Zunami tokens reflecting his share in total strategies holdings. Zunami tokens can be exchanged for any underlying strategies' asset using one of withdraw functions. It supports gas optimized deposit/withdraw methods via delegation to operator, who executes batch of multiple users requests.



## Issues

### C13-01 Default deposit/withdraw pid change concerns

● High

☑ Acknowledged

- a. The admin has to control the actual withdraw pid has sufficient **lpshares** amount to service users withdraw needs, otherwise users will be unable to return their assets;
- b. The admin has to control, while setting default withdraw/deposit pool, that addressed strategy has similar or close price of used liquidity token for staking, otherwise it will cause unfair distribution of **lpshares**.

```
function setDefaultDepositPid(uint256 _newPoolId) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_newPoolId < _poolInfo.length, 'Zunami: incorrect default deposit pool id');
    defaultDepositPid = _newPoolId;
    emit SetDefaultDepositPid(_newPoolId);
}

function setDefaultWithdrawPid(uint256 _newPoolId) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_newPoolId < _poolInfo.length, 'Zunami: incorrect default withdraw pool id');
    defaultWithdrawPid = _newPoolId;
    emit SetDefaultWithdrawPid(_newPoolId);
}
```

### Recommendation

Consider reworking mechanism of changing active PID to allow users either choose individually or update the default one if admin is not responding.

### C13-02 Lacks of validation on strategy duplication

● High

☑ Acknowledged

There is no check the strategy had not been already included when new pool is added. As once added strategy can't be removed, it will result to permanent **totalHoldings** miscalculation and consequently **lpshares** unfair distribution.

## Recommendation

Duplicated strategies must be denied in the `addPool()` function or holdings calculations should account duplicated strategies.

### C13-03 Excessive owner's rights

● High

✓ Acknowledged

a. If the admin account is compromised a malefactor can restrict all withdrawals by `setAvailableWithdrawalTypes()` function;

```
function setAvailableWithdrawalTypes(uint8 newAvailableWithdrawalTypes)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
{
    require(
        newAvailableWithdrawalTypes <= ALL_WITHDRAWAL_TYPES_MASK,
        'Zunami: wrong available withdrawal types'
    );
    availableWithdrawalTypes = newAvailableWithdrawalTypes;
}
```

b. If the admin account is compromised a malefactor can add an invalid strategy address, and since there is no functionality to remove the pool, this will absolutely break deposit functions, `totalHoldings` and `lpPrice` getters. It will also provoke UZD token unpeg possibility, as it relies on `Zunami.lpPrice()`;

```
function totalHoldings() public view returns (uint256) {
    ...
    totalHold += _poolInfo[pid].strategy.totalHoldings();
}

function lpPrice() external view returns (uint256) {
    return (totalHoldings() * 1e18) / totalSupply();
}
```

c. If the admin account is compromised a malefactor can withdraw all strategies' funds by adding a malicious strategy and transferring all funds to it with `moveFundsBatch()`.

## Recommendation

Consider adding method for removing invalid strategies.

All DEFAULT\_ADMIN\_ROLE bearers must be secured, preferably by Timelock and MultiSig contracts.

### C13-04 Fees may reach up to 100%

Medium

Resolved

During `autoCompoundAll()` and `moveFundsBatch()` calls, strategies swap the rewards they accumulated on `feeToken` and reinvest it. Strategies take the fee from received `feeToken` calculated with `calcManagementFee()`.

```
function calcManagementFee(uint256 amount) external view returns (uint256) {  
    return (amount * managementFee) / FEE_DENOMINATOR;  
}
```

The `managementFee` parameter is set by the admin and may be close to 100%, what will leave investors without further profit.

```
function setManagementFee(uint256 newManagementFee) external onlyRole(DEFAULT_ADMIN_ROLE)  
{  
    require(newManagementFee < FEE_DENOMINATOR, 'Zunami: wrong fee');  
    managementFee = newManagementFee;  
}
```

## Recommendation

Limit the max fee percent or justify your design in NatSpec description and transfer ownership to admin contract to avoid malicious manipulations over fees.

### C13-05 Events problems

Low

Partially fixed

a. No strategy or pool field in `Deposited()` event;

- b. No event is emitted in the `setManagementFee()` function;
- c. The same `FailedWithdrawal` event for different withdrawal failure reasons in the `completeWithdrawals()` function: event is emitted on low balance and strategy call fail;
- d. `FailedWithdrawal` is emitted alongside the `Withdrawn` event in the `completeWithdrawalsOptimized()` function.

## C13-06 Gas optimization

● Low

🔗 Partially fixed

- a. Unnecessary reads from storage of `tokens[i]` in the constructor section;
- b. Unchecked math could be used for `decimalsMultipliers[i]` calculations in the constructor section;
- c. Multiple reads from storage of `_poolInfo.length()` in the `claimAllManagementFee()`, `autoCompoundAll()` functions;
- d. Double read of `decimalsMultipliers[]` in the `completeDeposits()` function;
- e. Double read from storage of `_poolInfo[].lpShares`, `totalDeposited` in the `completeWithdrawals()` function;
- f. Multiple reads from storage of `_poolInfo[].lpShares`, `totalDeposited`, `tokens[]` in the `completeWithdrawalsOptimized()` function;
- g. Double read from storage of `_poolInfo[].lpShares`, `totalDeposited` in the `withdraw()` function;
- h. Unchecked math could be used in `AddedPool` event;
- i. Multiple reads of `_pendingDeposits[][]` in the `removePendingDeposit()` function.

## C13-07 Supported tokens constraints

● Info

☑ Acknowledged

Tokens with decimals greater than 18 are not supported.

## C13-08 Inconsistent constant variable usage

● Info

☑ Resolved

`POOL_ASSETS` constant should be used in the `delegateDeposit()`, `completeDeposits()`, `completeWithdrawalsOptimized()`, and `calcSharesAmount()` functions instead of 3 constant.

## C13-09 Spelling errors

● Info

☑ Resolved

Typos in `transferred` and `reciver`.

## C13-10 Pauser role proposal

● Info

☑ Acknowledged

Consider adding `PAUSER_ROLE` in order to implement automated monitoring without compromising the `DEFAULT_ADMIN` key, see `pause()` function:

```
function pause() external onlyRole(DEFAULT_ADMIN_ROLE) {
    _pause();
}
```

## 5. Conclusion

6 high, 4 medium, 15 low severity issues were found during the audit. 1 medium, 8 low issues were resolved in the update.

The reviewed contracts are highly dependent on the owner's account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of potential attacks.

## Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

## Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code



 [contact@hashex.org](mailto:contact@hashex.org)

 [@hashex\\_manager](https://t.me/hashex_manager)

 [blog.hashex.org](https://blog.hashex.org)

 [linkedin](https://www.linkedin.com/company/hashex)

 [github](https://github.com/hashex)

 [twitter](https://twitter.com/hashex)

**#HashEx**  
BLOCKCHAIN SECURITY