ackee blockchain

Zunami Audit

18 February 2022

by Ackee Blockchain

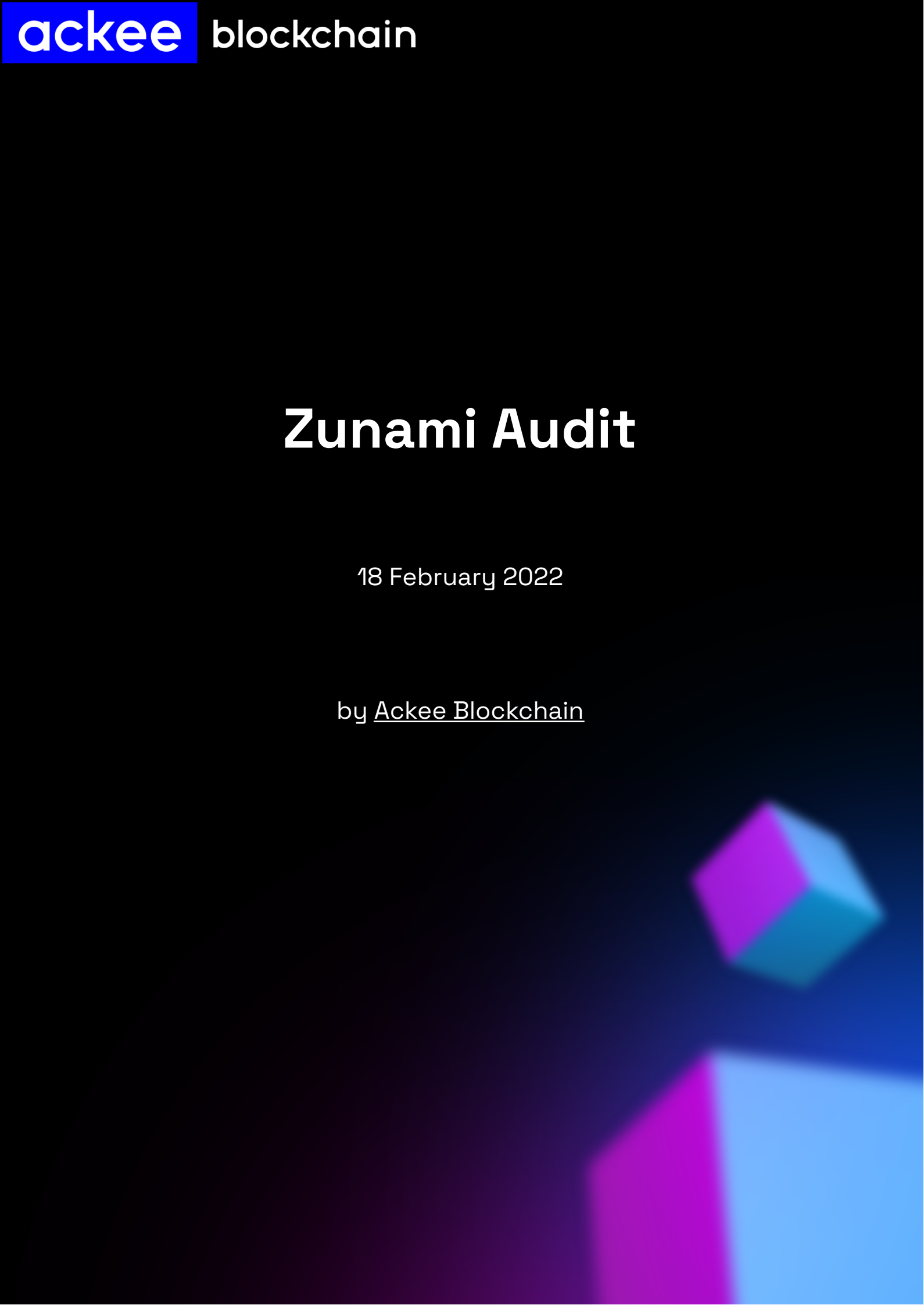

Table of Contents

Document Revisions

Revision	Date	Description
1.0	14 Jan 2022	Initial revision
1.1	18 Feb 2022	Updated issues

1. Overview

This document presents our findings in reviewed contracts.

1.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

1.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client, and the audit scope is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools is performed.
3. **Manual code review** is checked line by line for common vulnerabilities, code duplication, best practices, and the code architecture is reviewed.
4. **Local deployment + hacking** - contracts are deployed locally, and we try to attack the system and break it.
5. **Unit testing** - run unit tests to ensure that the system works as expected. Potentially we write our unit tests for specific suspicious scenarios.

1.3 Review team

The audit has been performed with a total time donation of 12 engineering days. The work has been divided between two auditors. The whole process was supervised by the Audit Supervisor.

Member's Name	Position
Štěpán Šonský	Lead Auditor
Lukáš Böhm	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

1.4 Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't be considered as a complete list of all existing issues.

2. Scope

This chapter describes the audit scope, contains provided specification, used documentation and set main objectives for the audit process.

2.1 Coverage

Files being audited:

- strategies
  - AaveCurveConvex.sol
  - BaseCurveConvex.sol
  - BaseCurveConvex2.sol
  - BaseCurveConvex4.sol
  - BUSDV2CurveConvex.sol
  - DUSDCurveConvex.sol
  - FraxCurveConvex.sol
  - IronBankCurveConvex.sol
  - LUSDCurveConvex.sol
  - MIMCurveConvex.sol
  - MUSDCurveConvex.sol
  - OUSDCurveConvex.sol
  - RSVCurveConvex.sol
  - SUSDCurveConvex.sol
  - USDKCurveConvex.sol
  - USDNCurveConvex.sol
  - USDPCurveConvex.sol
- utils
  - Constants.sol
- Zunami.sol

Sources revision used during the whole auditing process:

- Repository: https://github.com/ZunamiLab/ZunamiProtocol
- Commit:
  - **Rev. 1.0:** 37dccabf5aa3697dce5eaf6457debb3ac7404fdd
  - **Rev 1.1:** a444eb2cd040e73be02c80619739b9a1491a108f

Update rev. 1.1: Some of contracts has been refactored and renamed since previous audit, we summarize these changes bellow:

- `BaseStrat.sol` - New base contract
- `BaseCurveConvex.sol` -> `CurveConvexStrat.sol`
- `BaseCurveConvex2.sol` -> `CurveConvexStrat2.sol`
- `BaseCurveConvex4.sol` -> `CurveConvexStrat4.sol`

Because many code changes have happened since rev. 1.0., this document update should be considered as a revision rather than full re-audit. In the allocated 2 engineering days we focused primarily on previously reported issues rather than discovering new ones.

2.2 Supporting Documentation

We've used the official documentation, which is very brief in time of the auditing process and missing deep technical documentation.

- https://zunamilab.gitbook.io/product-docs/

"The return rates in stablecoin pools are highly volatile. Today, one of the pools shows the best APY / APR on the market, but it is already an outsider a week later. Finding the best pools and transferring funds from pool to pool is expensive and requires constant market research, which is not suitable for generating passive income.
The Zunami Protocol uses a couple of formulas to calculate APR / APY and select the most profitable pool when depositing funds. When a user wants to withdraw the funds, the most unprofitable pool is calculated, and funds are withdrawn from it. If funds are stagnant in a non-profitable strategy, the mechanism of manual rebalancing of funds is used, which is specified in the moveFunds() and moveFundsBatch() functions. The funds from the least profitable strategy are transferred to the best pool. There is also a mechanism for withdrawing funds in case of force majeure emergencyWithdraw()."

2.3 Objectives

We've defined the following main objectives of the audit:

- Check the activity on the GitHub repository.
- Review the code quality, architecture and best practices.
- Check for vulnerabilities if nobody is able to steal funds or damage contracts.
- Validate algorithms and math calculations for misbehaviors.
- Check if the contracts' owner is not overpowered.

3. System Overview

This chapter describes the audited system from our understanding. The whole project is based on the HardHat development framework.

We've identified 4 main contracts which are worth describing for complete understanding of the audited system.

3.1 Zunami.sol

Following chapter describes our detailed understanding of the `Zunami.sol` contract and its parts.

The contract uses OpenZeppelin library and inherits from `Context`, `Ownable` and `ERC20`, it also uses `SafeERC20` library for `IERC20Metadata`.

Structures

- PendingDeposit - contains `amounts` array and `depositor` address.
- PendingWithdrawal - contains `amounts` array and `withdrawer` address.
- PoolInfo - contains strategy and `startTime`.

Constants

- `POOL_ASSETS = 3`
- `FEE_DENOMINATOR = 1000`
- `MIN_LOCK_TIME = 86400`

Modifiers

- `onlyOwner` - Can be called only by the owner address
- `isLocked` - Can be called only when `isLock = false`

(rev. 1.1)

- `isStrategyStarted` - Can be called only when the strategy began

Constructor

The constructor has no parameters and initializes the ERC20 parent with "ZunamiLP" token name and "ZLP" symbol. Then it inits `tokens` array with values:

- `Constants.DAI_ADDRESS`
- `Constants.USDC_ADDRESS`
- `Constants.USDT_ADDRESS`

Functions

`setManagementFee(uint256 newManagementFee) external onlyOwner`

- Sets new management fee (percent * 10)
- Has to be less than `FEE_DENOMINATOR`

`calcManagementFee(uint256 amount) public view virtual`

- Calculates the management fee for specific amount

`totalHoldings() public view virtual returns (uint256)`

- Returns sum of holdings from all pools

`lpPrice() public view virtual returns (uint256)`

- Calculates LP token price as `totalHoldings() / totalSupply()`

`delegateDeposit(uint256[3] memory amounts) external virtual isLocked`

- Transfers amounts of tokens from the user to the contract
- Sets pending deposit amounts for the user into the `accDepositPending` mapping

`delegateWithdrawal(uint256 lpAmount, uint256[3] memory minAmounts) external virtual`

- Adds a new `pendingWithdrawal` into the `pendingWithdrawals` array

`completeDeposits(address[] memory userList, uint256 pid) external virtual onlyOwner`

- Normalizes amounts to 18 decimals
- Transfers `totalAmounts` of tokens into the `poolInfo[pid]`
- Completes pending deposits for addresses in the `userList`
- Increments `totalDeposited` by deposited amounts
- Calculates and mints LP token share to users
- Updates pools using `updateZunamiLpInStrat()`
- Clears `accDepositPending` values for `userList`

`completeWithdrawals(uint256 withdrawalsToComplete, uint256 pid) external virtual onlyOwner`

- Calls `delegatedWithdrawal()` for items in the `pendingWithdrawals` array and the pool with id = `pid`.
- Maximum count of withdrawals is limited by `withdrawalsToComplete` param

`deposit(uint256[3] memory amounts, uint256 pid) external virtual isLocked returns (uint256)`

- Checks whether the strategy has started
- Transfers `amounts` of tokens from the user to the pool with id = `pid`
- Calls `deposit()` on the pool's strategy
- Increments `totalDeposited` by deposited amounts
- Calculates and mints LP token share to the user
- Updates the pool

`withdraw(uint256 lpShares, uint256[3] memory minAmounts, uint256 pid) external virtual`

- Checks if the user has enough LP tokens
- Calls `withdraw()` on the pool's strategy
- Calculates user's deposit as `(totalDeposited*lpShares) / totalSupply()`
- Burns the user's LP token share
- Updates the pool
- Updates user's `deposited` value
- Updates `totalDeposited` value

`delegatedWithdrawal(address withdrawer, uint256 lpShares, uint256[3] memory minAmounts, uint256 pid) internal virtual`

- Checks if the user has enough balance of LP tokens
- Tries to withdraw `lpShares` from the pool `pid`
- Burns `lpShares` amount from the `withdrawer`
- Updates the pool
- Updates user's `deposited` value
- Updates `totalDeposited` value

`setLock(bool _lock) external virtual onlyOwner`

- Sets `isLock` value, which is used in the `isLocked()` modifier

`claimManagementFees(address strategyAddr) external virtual onlyOwner`

- Claims management fees from strategy on the address `strategyAddr`
- Transfers fees to the owner's address

`add(address _strategy) external virtual onlyOwner`

- Adds the strategy into the `poolInfo` array
- Sets its startTime to `block.timestamp + MIN_LOCK_TIME`

`moveFunds(uint256 _from, uint256 _to) external virtual onlyOwner`

- Withdraws all tokens from the pool `_from` to Zunami
- Transfers all tokens from the Zunami to the pool `_to`
- Calls `deposit()` on the `_to` pool's strategy
- Updates both pools

`moveFundsBatch(uint256[] memory _from, uint256 _to) external virtual onlyOwner`

- Withdraws all tokens from pools `_from` to Zunami
- Updates pools `_from`
- Transfers all tokens from Zunami to the pool `pid = 0`
- Updates the pool `_to`
- Tries to deposit amounts into the `pool[_to]`

`emergencyWithdraw() external virtual onlyOwner`

- Withdraws all funds from all pools except the pool 0
- Updates all pools except the pool 0
- Transfers all tokens from the Zunami to the pool 0
- Updates the pool 0
- Tries to deposit amounts into the pool 0

`pendingDepositRemove() external virtual`

- Transfers pending deposits back to the user
- Clears user's pending deposits

3.2 BaseCurveConvex.sol

Following chapter describes our detailed understanding of the `BaseCurveConvex.sol` strategy contract and its parts.

Constants

- `DENOMINATOR = 1e18`
- `USD_MULTIPLIER = 1e12`
- `DEPOSIT_DENOMINATOR = 10000`

Modifiers

- `onlyZunami` - Can be called only by the Zunami contract

Constructor

The constructor has 7 parameters:

- `address poolAddr`
- `address poolLPAddr`

- `address rewardsAddr`
- `uint256 poolPID`
- `address extraRewardsAddr`
- `address extraTokenAddr`
- `address extraTokenPairAddr`

It inits variables `pool, poolLP, extraToken, crv, crweth, wethcvx, wethusdt, extraPair, router, cvxPoolPID, booster, cvx, crvRewards, extraRewards` and fills `tokens` array with values `Constants.DAI_ADDRESS`, `Constants.USDC_ADDRESS` and `Constants.USDT_ADDRESS`.

Functions

`setZunami(address zunamiAddr) external onlyOwner`

- Creates `zunami` interface with the Zunami contract address

`getZunamiLpInStrat() external view virtual returns (uint256)`

- Returns `zunamiLpInStrat`

`totalHoldings() public view virtual returns (uint256)`

- Summarizes holdings in contract
- Calculates the LP price
- Calculates the CRV price
- Calculates the CVX price
- Adds up all and returns the result

`deposit(uint256[3] memory amounts) external virtual onlyZunami returns (uint256)`

- Normalizes amounts to 18 decimals
- Calculates minimal amounts for deposit
- Checks if minimal amounts are lower or equal than tokens in the pool
- Adds liquidity to Curve pool
- Gets Curve pool tokens and deposit them to the booster contract

`withdraw(address depositor, uint256 lpShares, uint256[3] memory minAmounts) external virtual onlyZunami returns (bool)`

- Calculates `depositedShare`
- Sells rewards using `sellCrvCvx()`
- Calculates complete user balances
- Calculates liquidity amounts
- Transfers tokens to the depositor (liquidity amounts + user balances - fee)

`claimManagementFees() external virtual onlyZunami`

- Compare `USDT` balance of strategy contract with `managementFees`

- Transfer lower one to the owner address
- Sets `managementFees` to zero

`sellCrvCvx() public virtual`

- Gets balances and approvals for rewards `cvx` and `crv`
- Swaps these tokens through the defined `path[]` on the Uniswap
- Updates `managementFees`

`withdrawAll() external virtual onlyZunami`

- Withdraws all `crvRewards` tokens and unwrap them back into the underlying LP tokens
- Calls `sellCrvCvrx()`
- Transfers all holdings to the msg.sender (Zunami contract)

`updateMinDepositAmount(uint256 _minDepositAmount) external onlyOwner`

- Checks if `_minDepositAmount > 0 && _minDepositAmount <= 10000`
- If so, updates `minDepositAmount`

`updateZunamiLpInStrat(uint256 _amount, bool _isMint) external onlyZunami`

- Adds `_amount` to `zunamiLpInStrat` if `_isMint` is true
- Else subtracts `_amount` from `zunamiLpInStrat`

3.2 BaseCurveConvex2.sol, BaseCurveConvex4.sol

BaseCurveConvex2 and BaseCurveConvex4 have pretty much the same base logic as BaseCurveConvex. Contain all functions from BaseCurveConvex plus these additional two functions.

Additional Functions

`sellToken() public virtual`

- Performs an exchange of tokens

`sellExtraToken() public virtual`

- Swaps the extra token trough defined `path[]` on the Uniswap
- `path[]` depends on whether the extra token is `WETH` or not
- Updates `managementFees`

4.Security Specification

This section specifies single roles and their relationships in terms of security in our understanding of the audited system.

4.1 Actors

This part describes actors of the system, their roles, and permissions.

Owner

Deploys contracts and has special privileges in Zunami and strategies.

In Zunami, the owner can set management fees, complete deposits, complete withdrawals, set deposit lock, claim management fees, add pools, move funds between pools, and emergency withdrawals.

The owner can set the Zunami address and update the minimum deposit amount in strategies.

Also, the owner can transfer the ownership of contracts to another address and renounce ownership.

Zunami

Zunami actor means Zunami.sol contract, which has special privileges in strategies contracts thanks to `onlyZunami` modifier. Including deposit, withdrawal, claim management fees, withdraw all and update LP token amount (`zunamiLpInStrat`) in the strategy.

Strategies

Strategies are contracts which manage deposits and withdrawals from/to external pools.

Pools

External Curve pools are black boxes in our trust model and Zunami developers have no control over their security.

Users

User role means any external Ethereum address which can interact with the Zunami protocol. In the Zunami contract they can read the management fee amount, total holdings and LP token price. Also, users can deposit, delegate deposit, withdraw, and delegate withdraw. In strategy contracts, users can read the amount of Zunami

LP in the strategy, read total holdings, sell CRV and CVX owned by the contract, sell token owned by the contract and sell extra token owned by the contract,

4.2 Trust model

We've identified that users need to put a lot of trust into the contract owner. Owner has total control over the funds flow between pools.

Renouncing the ownership by the owner can cause irreversible damage to the contract, and nobody will be able to execute onlyOwner methods mentioned above in the Owner actor paragraph.

Users also have to put trust in the Curve, but the security of this external platform is not under Zunami's responsibility.

5. Findings

This chapter shows detailed output of our analysis and testing.

5.1 General Comments

We've used static analysis tools to check common issues, which raised suspicion of a reentrancy bug in the `BaseCurveConvex2.sol` `withdraw()` and `withdrawAll()` functions, but after a deep investigation, we didn't identify it as vulnerable.

General code quality is sub-average. The code contains many bugs, unused variables, and best practices violations. The architecture of base strategy contracts should be more abstract. There are many code duplications.

5.2 Issues

Using our toolset, manual code review, and unit testing, we've identified the following issues.

Low

Low severity issues are more comments and recommendations rather than security issues. We provide hints on how to improve code readability and follow best practices. Further actions depend on the development team's decision.

ID	Description	Contract	Line	Status
L1	Inconsistent iteration statement syntax	Zunami.sol	128 130 ...	Fixed
L2	Hardcoded token index	BaseCurveConvex.sol	213 214 225 249	Partially fixed
		BaseCurveConvex2.sol	271 272 283 307 318 324 346	

			369	
		BaseCurveConvex4.sol	275 276 286 309 323 340 363 371	
L3	Confusing modifier naming	Zunami.sol	58	Fixed

L1: Inconsistent syntax does not look professional. There is no difference between i++ and ++i in the iteration statement syntax. We recommend using established standard i++.

L1 (rev. 1.1): Iteration statement syntax has been unified.

L2: Hardcoded indexes are not the best practice. Constant token addresses from the `Constants.sol` can be used instead.

L2 (rev. 1.1): Issue has been fixed in many places but still remains in `CurveConvexStrat2.sol` (L239, L250) and `CurveConvexStrat4.sol` (L226, L236).

L3: Modifier is named `isLocked()`, but it is actually `isNotLocked()`, because it requires the `isLock` variable to be `false`.

L3 (rev. 1.1): The modifier has been renamed.

Medium

Medium severity issues aren't security vulnerabilities, but should be clearly clarified or fixed.

ID	Description	Contract	Line	Status
M1	Unused virtual keyword	Zunami.sol	74 81 90 94 111 121	Fixed

			170 187 222 243 259 264 270 278 299 322 347	
M2	Public functions can be external	Zunami.sol	74 90	Fixed
		BaseCurveConvex.sol	95	
		BaseCurveConvex2.sol	102	
		BaseCurveConvex4.sol	97	
M3	State variable could be local	Zunami.sol	165	Fixed
M4	Missing const	Zunami.sol	40	Fixed
M5	Unused variables	Zunami.sol	45 46 47	Fixed
M6	Code duplication	BaseCurveConvex.sol BaseCurveConvex2.sol BaseCurveConvex4.sol		Fixed
M7	Interface issues	IZunami.sol	5 7 9	Partially fixed
M8	Unintended feature - Renounce ownership	All		Fixed
M9	Missing const	CurveConvexStrat.sol	23	Reported
		CurveConvexStrat2.sol	24	

		CurveConvexStrat.sol	23	

M1: Functions are not supposed to be overridden, `virtual` marks are useless.

M1 (rev. 1.1): `virtual` keyword has been removed in all occurrences.

M2: Functions are never called internally. (Gas saving)

M2 (rev. 1.1): All unnecessary `public` functions have been changed to `external`.

M3: Variable `userCompleteHoldings` is used in `completeDeposits()` only. It should be initialized locally in the function. (Gas saving)

M3 (rev. 1.1): The variable has been moved to function scope.

M4: Variable `FEE_DOMINATOR` is read-only; thus, it should be constant. (Gas saving)

M4 (rev. 1.1): Variable has been changed to constant.

M5: Unused variables should be deleted. (Gas saving)

M5 (rev. 1.1): Unused variables have been removed.

M6: There are many code duplicities in base strategies, which could be solved by using better inheritance. Functions `sellCrvCvx()` and `claimManagementFees()` are completely the same, and many other functions' parts are duplicated. We strongly recommend refactoring these contracts to decrease potential points of failure and increase readability.

M6 (rev. 1.1): Functions `sellCrvCvx()` and `claimManagementFees()` has been moved to the parent contract.

M7: Zunami.sol should inherit from the interface IZunami.sol, but it doesn't. In addition `IZunami.sol` has functions, which are missing in `Zunami.sol` and they are not called by anyone. These functions' definitions should be removed from the interface.

M7 (rev. 1.1): Unimplemented functions have been removed from the `IZunami` interface. However, `Zunami.sol` still doesn't inherit from `IZunami.sol`.

M8: Open Zeppelin's Ownable pattern allows the current owner to renounce the ownership of the contract (can happen even accidentally), which could potentially

cause the inability to call any function with the `onlyOwner` modifier. We recommend overriding the `renounceOwnership()` method to disable this unwanted feature.

M8 (rev. 1.1): `renounceOwnership()` function has been overridden to revert the transaction.

M9 (rev. 1.1): The new variable `usdtPoolId` has no write access across contracts. It should be constant.

High

High severity issues are potential security vulnerabilities, which require specific steps and conditions to be exploited. Or bugs in the contract logic which doesn't endanger user funds. These issues have to be fixed.

ID	Description	Contract	Line	Status
H1	Management fee rewriting	BaseCurveConvex.sol	250	Fixed
		BaseCurveConvex2.sol	347	
		BaseCurveConvex4.sol	310 341 364	

H1: Functions sellCrvCvx() and sellExtraToken() rewrite the value of `managementFees`. This leads to unintended behavior and management fees don't get accumulated. It creates lower withdrawal fees for users and fund loss for Zunami.

H1 (rev. 1.1): The issue has been fixed.

Critical

Direct critical security threats, which could be instantly misused to attack the system. Or critical bugs in the logic, which leads to contract misbehavior or users' funds loss. These issues have to be fixed.

ID	Description	Contract	Line	Status

C1	Bug in the logic - wrong pool id	Zunami.sol	313	Fixed
C2	Rewriting deposit amounts	Zunami.sol	105	Fixed

C1: In the function `moveFundsBatch()` where funds are moving from multiple pools to one, there is hardcoded `poolInfo[0].strategy` instead of `poolInfo[_to].strategy`. We've identified this as a critical bug in the contract logic, which leads to unintended behavior and a potential funds loss due to non-optimal transfers.

Also in case of parameter _to != 0 `updateZunamiLpInStrat()` doesn't get called on the `pool[0]`, so `zunamiLpInStrat` stays the same. On the other hand `updateZunamiLpInStrat()` gets called on `pool[_to]` with the amount which doesn't belong to the pool. This scenario leads to mathematical inconsistencies across pools, and users can potentially withdraw more funds than they are eligible to.

C1 (rev. 1.1): The issue has been fixed.

C2: If the user calls `delegateDeposit()` function multiple times in a sequence, then `accDepositPending[user]` gets rewritten. There should be an array iteration and `+=` operator on every amount item. This bug leads to the user's funds loss in case he calls `delegateDeposit()` multiple times before the owner call's `completeDeposits()`.

Example scenario:

1. User deposits 100 DAI, 100 USDC, 100 USDT using `delegateDeposit()`
2. Funds are transferred from the user's account into the Zunami contract
3. User deposits again 100 DAI, 100 USDC, 100 USDT using `delegateDeposit()`
4. Funds are transferred from the user's account into the Zunami contract
5. The owner calls `completeDeposits()`
6. The user receives LP token for $300, but he actually paid $600

C2 (rev. 1.1): The issue has been fixed.

5.3 Unit testing

Statements test coverage in `Zunami.sol` is nearly 99% and `BaseCurveConvex.sol`, `BaseCurveConvex2.sol` and `BaseCurveConvex4.sol` is 80-90%, which is solid, but we strongly recommend increasing test coverage as much as possible.

`npm run test:coverage` output:

```
  48 passing (1m)

--------------------------|----------|----------|----------|----------|----------------|
File                      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
--------------------------|----------|----------|----------|----------|----------------|
 contracts/               |    98.63 |    59.09 |      100 |    98.65 |                |
  Zunami.sol              |    98.63 |    59.09 |      100 |    98.65 |        247,248 |
 contracts/strategies/    |    84.46 |    54.17 |    81.48 |    84.39 |                |
  AaveCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  BUSDV2CurveConvex.sol   |      100 |      100 |        0 |      100 |                |
  BaseCurveConvex.sol     |    83.96 |    64.29 |      100 |    83.96 |... 249,250,253 |
  BaseCurveConvex2.sol    |    88.95 |    54.76 |      100 |    88.89 |... 369,370,373 |
  BaseCurveConvex4.sol    |    80.12 |       50 |      100 |       80 |... 356,363,364 |
  DUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  FraxCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  IronBankCurveConvex.sol |      100 |      100 |        0 |      100 |                |
  LUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  MIMCurveConvex.sol      |      100 |      100 |        0 |      100 |                |
  MUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  OUSDCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  RSVCurveConvex.sol      |      100 |      100 |        0 |      100 |                |
  SUSDCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  USDKCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  USDNCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  USDPCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
 contracts/utils/         |      100 |      100 |        0 |      100 |                |
  Constants.sol           |      100 |      100 |      100 |      100 |                |
  MockERC20.sol           |      100 |      100 |        0 |      100 |                |
--------------------------|----------|----------|----------|----------|----------------|
All files                 |    87.97 |    55.71 |    85.33 |    87.97 |                |
--------------------------|----------|----------|----------|----------|----------------|
```

Update rev. 1.1: Test count has been increased from 48 to 127 and all are passing. Test coverage has been improved as well.

```
--------------------------|----------|----------|----------|----------|----------------|
File                      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
--------------------------|----------|----------|----------|----------|----------------|
 contracts/               |     98.8 |       62 |    95.45 |     97.7 |                |
  Zunami.sol              |     98.8 |       62 |    95.45 |     97.7 |259,260,261,489 |
 contracts/strategies/    |    94.36 |    64.86 |    76.09 |    94.38 |                |
  AaveCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  BUSDV2CurveConvex.sol   |      100 |      100 |        0 |      100 |                |
  BaseStrat.sol           |    97.78 |    64.29 |       90 |    97.83 |            162 |
  CurveConvexStrat.sol    |    91.55 |       60 |      100 |    91.55 |... ,99,133,154 |
  CurveConvexStrat2.sol   |    97.37 |    73.08 |      100 |    97.37 |     75,164,186 |
  CurveConvexStrat4.sol   |    91.59 |    58.33 |      100 |    91.59 |... 117,161,186 |
  DUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  FraxCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  IronBankCurveConvex.sol |      100 |      100 |        0 |      100 |                |
  LUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  MIMCurveConvex.sol      |      100 |      100 |        0 |      100 |                |
  MUSDCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  OUSDCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  RSVCurveConvex.sol      |      100 |      100 |        0 |      100 |                |
  SUSDCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
  USDKCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  USDNCurveConvex.sol     |      100 |      100 |        0 |      100 |                |
  USDPCurveConvex.sol     |      100 |      100 |      100 |      100 |                |
 contracts/utils/         |      100 |      100 |        0 |      100 |                |
  Constants.sol           |      100 |      100 |      100 |      100 |                |
  MockERC20.sol           |      100 |      100 |        0 |      100 |                |
--------------------------|----------|----------|----------|----------|----------------|
All files                 |    95.83 |    63.71 |    81.16 |    95.51 |                |
--------------------------|----------|----------|----------|----------|----------------|
```

There are 48 unit tests in `Zunami.test.ts` and all passing. Tests are using mainnet forking for external pools, which is a good practice and a much better approach than using custom mocks. However, we strongly suggest covering scenarios related to our reported issues.

`npx hardhat test` output:

Solc version: 0.8.4	Optimizer enabled: true	Runs: 200	Block limit: 40000000 gas	
Methods				
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
AaveCurveConvex	setZunami	–	–	46156	1	–
AaveCurveConvex	updateMinDepositAmount	–	–	28682	2	–
ERC20	approve	46158	59975	51565	15	–
ERC20	transfer	51818	65625	60211	15	–
OUSDCurveConvex	setZunami	–	–	46083	1	–
OUSDCurveConvex	updateMinDepositAmount	–	–	28682	1	–
SUSDCurveConvex	setZunami	–	–	46150	1	–
SUSDCurveConvex	updateMinDepositAmount	–	–	28749	1	–
USDPCurveConvex	setZunami	–	–	46083	1	–
USDPCurveConvex	updateMinDepositAmount	–	–	28682	1	–
Zunami	add	73500	90624	77793	4	–
Zunami	claimManagementFees	59093	60171	59737	5	–
Zunami	completeDeposits	1958228	2368236	2127966	5	–
Zunami	completeWithdrawals	1424241	2918890	2278554	4	–
Zunami	delegateDeposit	182714	253914	202170	12	–
Zunami	delegateWithdrawal	79122	96222	85962	10	–
Zunami	deposit	1500015	2486457	1812019	10	–
Zunami	emergencyWithdraw	–	–	3695787	1	–
Zunami	moveFunds	2208384	3288164	2748274	2	–
Zunami	moveFundsBatch	–	–	3889326	1	–
Zunami	pendingDepositRemove	–	–	161258	2	–
Zunami	setLock	23944	45856	34900	2	–
Zunami	setManagementFee	–	–	30801	1	–
Zunami	withdraw	1392128	1675082	1445122	10	–
Deployments					**% of limit**	
AaveCurveConvex		–	–	3302755	8.3 %	–
OUSDCurveConvex		–	–	4758271	11.9 %	–
SUSDCurveConvex		–	–	4412110	11 %	–
USDPCurveConvex		–	–	4758591	11.9 %	–
Zunami		–	–	3938212	9.8 %	–

48 passing (10m)

6. Conclusion

At the beginning of the auditing process, we had to request technical documentation of the project due to missing documentation and comments in the code. Developers prepared brief documentation with mathematical formulas in a few days. However, it doesn't describe every single function and its intended behavior. Meanwhile, the lead auditor defined the audit methodology and objectives.

We performed a static analysis using Slither, which raised false-positive reentrancy. We've identified many issues across our severity scale during our intense manual code review. We have found two critical issues C1 and C2. Both are serious bugs in the contract logic which lead to a loss of funds. High severity issue H1 also has a big impact on the monetary model. We also strongly recommend addressing all medium issues, and regarding low issues, we let developers decide whether to fix them or not.

We've rated the code quality as sub-average because the code contains many best practices violations. We've found many bugs in contracts' logic, so there is still a probability that there are more of them.

We can't recommend deploying contracts in the current state since the system does not behave as described in the documentation and endanger users' funds.

Update rev. 1.1: Developers updated the code regarding our findings across the severity spectrum. Critical and high issues have been fixed, medium and low recommendations have been applied.

The structure of contracts has been slightly changed, but the logic remains almost the same. One new base strategy contract has been created from which other strategies inherit. The codebase has become more readable and looks more professional also due to the added NatSpec format documentation. However, one new minor issue (M9) appeared.

ackee blockchain

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq