HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Nitro Cartel
Date: May 31, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Nitro Cartel
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	Index Protocol
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://nitrocartel.finance/
Changelog	04.04.2023 - First Review 03.05.2023 - Second Review 31.05.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Nitro Cartel (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/nitroarthur/arbitrove-smart-contracts/
Commit	85abc9e
Whitepaper	-
Functional Requirements	Arbitrove Docs
Technical Requirements	Repository README
Contracts	File: ./src/contracts/AddressRegistry.sol SHA3: 89a6e6fe00355c9489ad8b0c618cc87611fca3386706ae134621662b67bc2fc2 File: ./src/contracts/vault/FeeOracle.sol SHA3: f85a5de61663489c42b878a7f438d87b17a233dadb32821a6d54d87f24256690 File: ./src/contracts/vault/Vault.sol SHA3: 785127afe8a99c2e4ab74ee5fac7b0e089d0a5f366822ff231e8c67b601f1f94 File: ./src/contracts/Router.vy SHA3: 69c40756254abb00de56e0da45a894b3a823fd6ca7793f973f79982fb23473b7 File: ./src/contracts/strategy/IStrategy.sol SHA3: baa091dceb6552a91d83c927a47dd7adbe0b31c58eb99634b972298c0edbb793 File: ./src/contracts/vault/IVault.sol SHA3: 594fed31ed78280f7fd188a777f3a3e392d2569f283148f222526e48b0b5c76f

Second review scope

Repository	https://github.com/nitroarthur/arbitrove-smart-contracts/
Commit	7927f31
Whitepaper	-
Functional Requirements	Arbitrove Docs

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Technical Requirements	Repository README.md
Contracts	File: ./src/contracts/AddressRegistry.sol SHA3: 09239082710aed534e7ddf02c308f59fcfd9474d512c50321846c4ad9260e761 File: ./src/contracts/strategy/IStrategy.sol SHA3: 25d631964ee1931d248f796153cf0bdcefaad0085e2fa5e9059bf0d42b3964aa File: ./src/contracts/vault/FeeOracle.sol SHA3: fdee75cee758a2a419bc8fbb7571cbbce08b6e863ef74b85adf163c059715df6 File: ./src/contracts/vault/IVault.sol SHA3: 06a37a5d2038301fbebf18c39a20d4a6008b40619278e1fe82b8a7c8a8481bff File: ./src/contracts/vault/Vault.sol SHA3: 0199b55258b225abb6344492c9a300e400236db6456503d37eee6a26ac6f8cb0 File: ./src/contracts/Router.vy SHA3: c1a943247e4d9aeb1fa837235a8c3e7a6b53bf8c776372b9d5184db7fbac8373

Third review scope

Repository	https://github.com/nitroarthur/arbitrove-smart-contracts/
Commit	3bb9ee7
Whitepaper	-
Functional Requirements	Arbitrove Docs
Technical Requirements	Repository README.md
Contracts	File: ./src/contracts/AddressRegistry.sol SHA3: 2b5382325521bfe9d5f1f799bc668b921df8c39e3a3cee42c8c4e2cc37ebcc6a File: ./src/contracts/strategy/IStrategy.sol SHA3: 25d631964ee1931d248f796153cf0bdcefaad0085e2fa5e9059bf0d42b3964aa File: ./src/contracts/vault/FeeOracle.sol SHA3: 1b0bbc4d73756b82094e23114bc1253c1e9aa50f6da1cb506a2371c311423f2b File: ./src/contracts/vault/IVault.sol SHA3: 97d08f44188b4a757ac104511a076ce3438d88499d0d83e75e73d19394d6ee22 File: ./src/contracts/vault/Vault.sol SHA3: 8a6a2adf2c2e7689dd1df54ba9ff2ddb3c8476fb0cc17a3aa19e70c7753246af File: ./src/contracts/IRouter.sol SHA3: 2cadcac27fb5d4266f2de6d4893a66a9f97f012919b82a0fee2dbe03d5a56f34 File: ./src/contracts/Router.vy SHA3: dd83c905358eeacb52c04c4066b0d26b3e6698225355de2f056fdee741208b8c

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements provided.
- Technical documentation is provided but limited.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Small style guide violations.

Test coverage

Code coverage of the project is **100%** (branch coverage).

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**. The system users should acknowledge all the risks summed up in the risks section of the report.

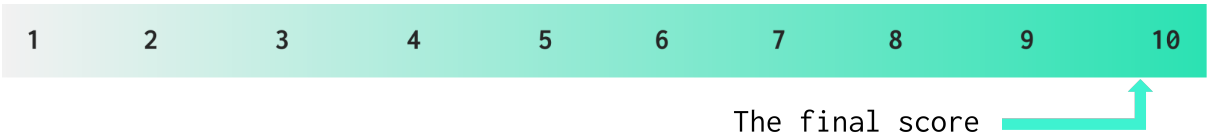

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
04 April 2023	22	22	8	2
03 May 2023	2	5	4	0
31 May 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Risks

- The system is highly centralized; each privilege role, if compromised, can lead to a loss of user funds.
- The Dark Oracle role used to provide token prices in the Router.vy is highly privileged; all systems behind this role are not part of this audit. There are no guarantees that said oracle behaves as expected and provided tokens prices in the `processMintRequest()` and `processBurnRequest()` functions will be correct.
- Many of the key elements of the system (vault, oracles, strategies) can be changed by the owner at any time.
- Strategies and tokens used by the protocol cannot be validated, as they are outside the scope of the audit.
- In the case of a Dark Oracle account compromise, an attacker could interact with the router and provide incorrect token prices.
- The function `getAmountAcrossStrategies()` in Vault.sol calls the method `getComponentAmount()`, which is outside the audit scope. The method may change in the future, as every new implemented strategy can have a different `getComponentAmount()` function. It is not possible to verify the logic of that call and its potential vulnerabilities.
- There is no valid on-chain mechanism for rebalancing the funds. There is only an `rebalance()` owner function to retrieve funds from the contract.
- The system integrates a Rebalancer contract, which is out of the audit scope.
- There is no withdrawal mechanism from the strategies, and the flow of funds is unknown after the approval of funds passed to the strategy. It may be the case that there are no funds in the Vault to perform the `withdraw()` function.
- The fee in the Router and FeeOracle contracts is limited not to be bigger than the 50%, which is still big enough value.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

Arbitrove Protocol is a yield-bearing index protocol that allows people to one-click mint an index that gives exposure to a batch of strategies consisting of yield-bearing assets. Unlike traditional index protocols that only hold tokens, Arbitrove Protocol dynamically deploys capital to strategies.

The contracts in scope are:

- **Router.vy** - entry point for users to interact with the Vault.
- **Vault.sol** - facilitates the deposit and withdrawal of funds, and helps manage assets across different strategies. Interactions with this contract are sent through the Router.
- **AddressRegistry.sol** - manages the mapping of strategies to supported coins.
- **FeeOracle.sol** - implements a fee oracle that provides deposit and withdrawal fees to be used by the Vault contract. The fees are based on the current weight of a coin in the Vault compared to its target weight.

Privileged roles

- Router: it is the entry point for users to interact with the Vault.
- DarkOracle: oracle used in Router to get price and input parameters, as well as having access control privileges to some functions.
- Owner: set in the initialization functions of the contracts. Has admin privileges to update management state variables.
- User: can interact with the system to deposit funds in exchange of an interest-bearing indexed token, and vice-versa.

Recommendations

- Increase test coverage to 100%.
- Provide documentation (or code) for dark oracle (and strategies, if possible).
- Use multi-signature ⅗ wallets for all privileged roles in the system.
- Update public documentation about all privileged roles and their functions, and their impact on the protocol. Describe the rebalancing mechanism and the approval process for the strategy.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Not Relevant
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Data Consistency

The *approveStrategy()* function contains a check:

require(i==strategies.length)

This check is incorrect since it will cause a *revert* for all the cases in which the *strategy* to *approve* exists in the strategies array.

This leads to situations where strategy cannot be added to the Vault.

Path:
./src/contracts/vault/Vault.sol : approveStrategy()

Recommendation: It is recommended to change the operator from *==* to *!=* in order to include all possible positions of the *strategy* in the *strategies* array, while still performing as expected.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

C02. Access Control Violation

The *cancelMintRequest()* function can be executed by any address in case the mint request is expired, but it also allows to specify a *refund* parameter, meaning that anyone can cancel expired requests without refund, leading to funds loss.

Path:
./src/contracts/Router.vy : cancelMintRequest()

Recommendation: Do not allow canceling requests without a refund option for not authorized users (only for oracle). Document this behavior.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

■■■ High

H01. Data Consistency

The function *removeStrategy()* resets the *strategyWhitelist* to *0* but does not remove the strategy from the coins in *coinToStrategy[]* array.

Since the array of strategies for each coin is not updated, it can be the case that:

1. The variable *strategyWhitelist* is reset so that no coin can use it.
2. The function *addStrategy()* is called later in order to re-add the same strategy, expecting to add it only for the inputted coins.
3. Previous coins containing the strategy will still include it, even if it is not supposed to happen, leading to unexpected behavior.

Path:
./src/contracts/AddressRegistry.sol : removeStrategy()

Recommendation: Consider removing the strategy from the state variable *coinToStrategy* when calling the function *removeStrategy()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 3bb9ee7)

H02. Denial of Service

Denial of service – is a very common type of issue and attack. It can be executed in multiple ways. The issue mainly leads to a contract block and prevents further interactions. It does not necessarily bring an advantage to an attacker. Sometimes happens without an intended attack from a third party.

The *processBurnRequest()* function sends native tokens to *br.requester*, but the receiver could be a smart contract without payable receive function, so it would not be possible to send native tokens back.

This leads to the blocking of burn requests processing while the latest request would expire and be canceled without a refund option.

Path:
./src/contracts/Router.vy : processBurnRequest()

Recommendation: Add the possibility to process burn requests without sending native tokens to the requester. Add a separate function, so the requester is able to request funds back.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

H03. Highly Permissive Role; Data Consistency

Owner of the *Router* contract is able to reinitialize the contract, by setting new *vault*, *darkOracle,* and *addressRegistry* addresses.

This could lead to changing of the *vault* token after contract deployment, so a user would receive a refund with a not originally deposited token.

Path:
./src/contracts/Router.vy : reinitialize()

Recommendation: Document, or disable the possibility of changing mentioned addresses after the initialization.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

H04. Highly Permissive Role; Assets Integrity

Router contract has *rescueStuckTokens()* and *suicide()* functions to rescue ERC20 and native tokens, but these functions allow an owner to withdraw user funds as well, without previous notice.

Additionally, the presence of the *suicide()* function is dangerous and can lead to funds being locked in the contract.

During the second review issue is not fixed:

1. It is possible to *suicide()* contract when min and burn queues are not empty, which leads to funds loss.
2. It is possible to *suicide()* contract when some user's ERC20 tokens are still in the contract.

Path:
./src/contracts/Router.vy : rescueStuckTokens(), suicide()

Recommendation:

- Do not allow *suicide()* function execution when *mintQueue* and *burnQueue* are not empty and there ERC20 tokens on the contract balance. Consider removing the *suicide()* function.
- Track user's funds in a separate mapping after *submitBurnRequest()* and *submitMintRequest()* executions and allow *rescueStuckTokens()* to access only overbalanced funds.
- Implement a function to rescue the ETH from the contract, with restriction to only withdraw ETH that are not part of the users' actions.

Found in: 85abc9e

Status: Fixed (Revised commit: 3bb9ee7)

H05. Data Consistency

The function *getDistance()* performs a difference between *targetWeight* and *comparedWeight*.

In the case when *targetWeight == 0* (when the weight for the specific coin will be set by the admin to 0, for example Index with: 80% ETH, 20% USDC, 0% ARB) a *0* division will happen, causing a malfunction of this method and the related ones.

Path:
./src/contracts/vault/FeeOracle.sol : getDistance()

Recommendation: It is recommended to redesign the *getDistance()* function to be able to work with *0* as a *targetWeight.*

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

H06. Data Consistency; Race Condition

The current implementation of the queue in the *Router* contract operates on a "First-In, Last-Out" principle, which results in higher prioritization for the latest incoming requests compared to existing ones.

Under high load conditions, this approach may cause the initial requests to expire without being processed.

To ensure equitable and predictable queue processing, it is recommended to adopt a "First-In, First-Out" principle.

Path:
./src/contracts/Router.vy: processBurnRequest(), processMintRequest()

Recommendation: Process first requests from queue, instead of last (*pop()*).

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

H07. Data Consistency; Highly Permissive Role

The owner of the *FeeOracle* contract can use *setTargets()* at any time, changing coin weights. This operation disbalances pools, which gives an opportunity to arbitrageurs to have extra income on balancing pools and draining bonuses, which is unfair for existing depositors.

Any bad actor after the admin changes the coin weights will be able to easily extract part of the TVL from other users by leveraging the deposit/withdraw "bonus" mechanism if the maxBonus value is greater than 0.

The fee mechanism has no side effects.

We recommend rebalancing pools as a part of *setTargets()* execution, to prevent bonuses manipulations.

Path:
./src/contracts/vault/FeeOracle.sol: setTargets()

Recommendation: Implement rebalancing logic that is triggered with *setTargets* call. Removal of the "bonuses" mechanism on deposits or withdrawals will prevent uncontrolled extraction of the Vault TVL.

Found in: 85abc9e

Status: Mitigated (Bonus logic was mitigated by setting the maxBonus constant value to 0; thus, there are no risks for users after changing targets.)

H08. Highly Permissive Role; Assets Integrity

The *rebalance()* function of the *Vault* contract allows the owner to send user funds to any address. The function name is contradictory, because it expected to rebalance the vault, not just withdraw funds.

Behavior of rebalancing is not documented.

Path:
./src/contracts/vault/Vault.sol : rebalance()

Recommendation: Provide documentation of rebalancing logic. Consider implementing rebalancing as on-chain action.

Found in: 85abc9e

Status: Mitigated (Behavior is documented on client's website)

H09. Invalid Calculations; Requirements Violation

Multiple calculations in the contract are broken and lead to unexpected results.

FeeOracle contract calculates fee using logic: *fee = int256(deterioration * maxFee) / int256(weightDenominator)*, where *maxFee* is limited to 100, *deterioration* is a number less than 1e18, and *weightDenominator* is a 1e18 constant. Due to the rounding, the *fee* could be zero.

Vault contract uses this fee, but violates formulas provided in comments: instead of *(100 - fee) / 100 (fee denominator)* as per comment, it uses 1e18 as fee denominator.

Path:
./src/contracts/vault/FeeOracle.sol : getDepositFee(), getWithdrawalFee()

Recommendation: Recheck mentioned calculations and implement fixes. Consider using maxFee with 1e18 limitation.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

Medium

M01. Missing Event for Critical Value Update

The function *init()* does not emit relevant *events* when setting the state variables *feeOracle* and *router*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/AddressRegistry.sol : init()

Recommendation: Consider emitting an *event* in the function *init()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M02. Missing Event for Critical Value Update

The function *init829()* does not emit relevant *events* when setting the state variable *addressRegistry*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol : init829()

Recommendation: Consider emitting an *event* in the function *init829()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M03. Missing Event for Critical Value Update

The function *setPoolRatioDenominator()* does not emit relevant *events* when setting the state variables *poolRatioDenominator*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol : setPoolRatioDenominator()

Recommendation: Consider emitting an *event* in the function *setPoolRatioDenominator()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M04. Missing Event for Critical Value Update

The function *setCoinCapUSD()* does not emit relevant *events* when setting the state variables *coinCap[coin]*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol : setCoinCapUSD()

Recommendation: Consider emitting an *event* in the function *setCoinCapUSD()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M05. Missing Event for Critical Value Update

The function *setBlockCap()* does not emit relevant *events* when setting the state variables *blockCapUSD*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol : setBlockCap()

Recommendation: Consider emitting an *event* in the function *setBlockCap()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M06. Missing Event for Critical Value Update

The function *depositETHToStrategy()* does not emit relevant *events* when making a transfer of funds.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol : depositETHToStrategy()

Recommendation: Consider emitting an *event* in the function *depositETHToStrategy()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M07. Inefficient Gas Model: Loop of Storage Interactions

The function *getCoinToStrategy()* performs two loops, whose length is computed as *coinToStrategy[coin].length*.

The *storage* variable is accessed at every iteration, consuming a lot of Gas.

Instead, a new *memory* variable can be created to cache the *length*, which can then be used to compute the maximum value of *i*.

Path:
./src/contracts/AddressRegistry.sol : getCoinToStrategy()

Recommendation: Consider caching *coinToStrategy[coin].length* into a new memory variable, to be used to calculate the loop length.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M08. Inefficient Gas Model: Storage Abuse

The function *getCoinToStrategy()* performs two loops that access storage. The number of iterations of the loop is uncontrolled as it depends on the length of the state variable *coinToStrategy[coin]*.

Eventually, the state variable can be long enough to reach the block Gas limit.

The code can be optimized in terms of Gas by creating a new memory array of strategies from coinToStrategy[coin], and using it for the checks:

strategyWhitelist[coinToStrategy[coin][i]] < block.timestamp && strategyWhitelist[coinToStrategy[coin][i]] != 0

Path:
./src/contracts/AddressRegistry.sol: getCoinToStrategy()

Recommendation: Consider following the proposed optimization.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M09. Best Practice Violation: Unchecked Transfer

The function *claimDebt()* does not use the *SafeERC20* library for checking the result of ERC20 token *transfers*.

Tokens may not follow ERC20 standard and return *false* in case of *transfer* failure or not returning any value at all.

This can lead to unexpected behavior.

Path:
./src/contracts/vault/Vault.sol : claimDebt()

Recommendation: It is recommended to use the SafeERC20 library to interact with coins safely.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M10. Best Practice Violation: Unchecked Transfer

The function *rebalance()* does not use the *SafeERC20* library for checking the result of ERC20 token *transfers*.

Tokens may not follow ERC20 standard and return *false* in case of *transfer* failure or not returning any value at all.

This can lead to unexpected behavior.

Path:
./src/contracts/vault/Vault.sol : rebalance()

Recommendation: It is recommended to use the SafeERC20 library to interact with coins safely.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M11. State Variables Should Be Declared Constant

Changing *poolRatioDenominator* value can lead to unexpected contract behavior, so must be moved to constants and not changed after the contract deployment.

Path:
./src/contracts/vault/Vault.sol : setPoolRatioDenominator()

Recommendation: Make *poolRatioDenominator* as a constant (the best value should be 1e18).

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M12. Requirements Violation

In the *FeeOracle* in the *getCoinWeights* function, there is a comment *"// verify signature"*, but there is no signature validation, which is treated as a requirements violation or sign of non-finalized code.

Path:
./src/contracts/vault/FeeOracle.sol : getCoinWeights()

Recommendation: Verify that the code is complete, implement signature verification, or remove the comment.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M13. Missing Event for Critical Value Update

The function *cancelMintRequest()* does not emit relevant *events* when making a *transfer* of funds and updating *mintQueue*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: cancelMintRequest()

Recommendation: Consider emitting an *event* in the function *cancelMintRequest()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M14. Missing Event for Critical Value Update

The function *initialize()* does not emit relevant *events* when updating critical variables.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: initialize()

Recommendation: Consider emitting an *event* in the function *initialize()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M15. Missing Event for Critical Value Update

The function *reinitialize()* does not emit relevant *events* when updating critical variables.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: reinitialize()

Recommendation: Consider emitting an *event* in the function *reinitialize()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M16. Missing Event for Critical Value Update

The function *refundBurnRequest()* does not emit relevant *events* when updating *burnQueue* and performing a *transfer*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: refundBurnRequest()

Recommendation: Consider emitting an *event* in the function *refundBurnRequest()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M17. Missing Event for Critical Value Update

The function *suicide()* does not emit relevant *events* when using *selfdestruct*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: suicide()

Recommendation: Consider emitting an *event* in the function *suicide()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M18. Missing Event for Critical Value Update

The function *init()* does not emit relevant *events* when setting critical state variables.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/FeeOracle.sol: init()

Recommendation: Consider emitting an *event* in the function *init()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M19. Missing Event for Critical Value Update

The function *rebalance()* does not emit relevant *events* when sending native chain coins.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/vault/Vault.sol: rebalance()

Recommendation: Consider emitting an *event* in the function *rebalance()*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M20. Division Before Multiplication

The functions *deposit()* and *withdraw()* perform a division before multiplication since first calculate *poolRatio* from a division with *tvlUSD1e18X*, and later calculate the amount to burn or mint with another division.

This will result in imprecision in calculations.

Path:
./src/contracts/vault/Vault.sol: deposit(), withdraw()

Recommendation: It is recommended to perform divisions after multiplications.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M21. Unchecked Transfer

The *default_return_value* parameter can be used to handle ERC20 tokens affected by the missing return value bug in a way similar to OpenZeppelin's *safeTransfer* for Solidity.

Path:
./src/contracts/Router.vy: rescueStuckTokens(), processMintRequest(), cancelMintRequest(), processBurnRequest(), refundBurnRequest()

Recommendation: Use *default_return_value=True* for ERC20 token transfers.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M22. Inefficient Gas Model: Storage Abuse

In the *getCoinWeights()* function, a local *memory* variable *_targetsLength* is defined from the *storage* variable *targetsLength*.

However, *targetsLength* is already used before this declaration, not benefiting from the Gas saving.

Path:
./src/contracts/vault/FeeOracle.sol: getCoinWeights()

Recommendation: Consider defining *_targetsLength* before, so that it can be used in the following cases:

```
weights = new CoinWeight[](_targetsLength);
```

```
require(params.cpu.length == _targetsLength, "Oracle length error");
```

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

M23. Requirements Violation

According to the requirements - *supportedCoinAddresses* should contain addresses of coins that are linked to strategies, but this requirement is not met because of unreachable code.

During *addStrategy()* function execution the *uint256 supportedCoinLength = supportedCoinAddresses.length;* will be zero the first time the function is called, so loop *for (j = 0; j < supportedCoinLength; j++)* will be unreachable and no coins would be added to the *supportedCoinAddresses*.

As an additional consequence, clean-up of *supportedCoinAddresses* in *removeStrategy* is not possible.

The for loop body is also invalid, inside the loop, all *supportedCoinAddresses* should be checked and only when all of them are *!= coins[i]* the new coin should be added.

Path:
./src/contracts/AddressRegistry.sol: addStrategy(), removeStrategy()

Recommendation: Redesign the addStrategy() logic to allow the proper addition of strategies when supportedCoinAddress.length is 0. Fix the invalid logic of the for loop body.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

M24. Missing Event for Critical Value Update

The function *setFee()* does not emit relevant *events* when updating *fee*.

The function *setFeeDenominator()* does not emit relevant *events* when updating *feeDenominator*.

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Path:
./src/contracts/Router.vy: setFee(), setFeeDenominator()

Recommendation: Consider emitting an *event* in the *setFeeDenominator()* and *setFee()* functions.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

M25. Fee Is Not Limited

Consider limiting the *fee* and *feeDenominator* values in order to prevent unexpectedly high fees. In the current implementation, fee can be up to 100%.

Path:
./src/contracts/Router.vy: setFee(), setFeeDenominator()

Recommendation: Provide conscious limits for stored configuration values. Consider making *feeDenominator* as a constant.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

M26. Fee Is Not Limited

Consider limiting the *maxFee* value in order to prevent unexpectedly high fees. In the current implementation, fee can be up to 100%.

Path:
./src/contracts/vault/FeeOracle.sol: setMaxFee()

Recommendation: Provide conscious limits for stored configuration values.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

M27. Invalid Validation

During *deposit* to the *Vault* invalid validation:

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

getAmountAcrossStrategies(coin) + params._amount < coinCap[coin]

is performed.

According to the provided NatSpec, coinCap[coin] is a USD value of the cap, where getAmountAcrossStrategies(coin) and params._amount are the balances.

The two not related values are compared.

Path:
./src/contracts/vault/Vault.sol: deposit()

Recommendation: Multiply balances with coin price and compare it with coinCap.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

Low

L01. Style Guide: Order of Functions

The provided projects should follow the official guidelines. Functions should be grouped according to their *visibility* and ordered:

1. Constructor
2. Receive function (if exists)
3. Fallback function (if exists)
4. External
5. Public
6. Internal
7. Private

Path:
./src/contracts/Router.vy

Recommendation: Follow the official Solidity guidelines.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L02. Style Guide: Event Names

The Solidity Style Guide recommends the naming of *events* as *CapWords*.

Paths:
./src/contracts/vault/FeeOracle.sol : SET_TARGETS
./src/contracts/AddressRegistry.sol : SET_ROUTER, ADD_STRATEGY, ADD_REBALANCER, REMOVE_STRATEGY, REMOVE_REBALANCER
./src/contracts/vault/Vault.sol : SET_ADDRESS_REGISTRY

Recommendation: Follow the official Solidity guidelines.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L03. Style Guide: Missing NatSpec

There is a lack of NatSpec comments across the project's contracts, decreasing code interpretation, future upgrades and integrations.

Path:
./src/contracts/AddressRegistry.sol

Recommendation: Follow the official Solidity guidelines.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L04. Unindexed Events

Having indexed parameters in events makes it easier to search for these events, using indexed parameters as filters.

Paths:
./src/contracts/AddressRegistry.sol
./src/contracts/vault/Vault.sol
./src/contracts/Router.vy
./contracts/vault/FeeOracle.sol

Recommendation: Consider using the "indexed" keyword in events' parameters.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L05. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of *0x0*. This can lead to unwanted external calls to *0x0*.

Paths:
./src/contracts/vault/Vault.sol: setCoinCapUSD(), approveStrategy(), rebalance()
./src/contracts/Router.vy : initialize(), reinitialize()

Recommendation: Add zero address checks.

Status: Fixed (Revised commit: 7927f31)

L06. Redundant Operation; Gas Optimization

Existing implementation of *isNormalizedWeightArray* has a redundant operation and could be simplified, to improve readability and reduce Gas usage.

Path:
./src/contracts/vault/FeeOracle.sol : isNormalizedWeightArray()

Recommendation: Remove *j* variable. Replace *100 - j* with *100 - weights.length*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L07. State Variables Can Be Declared Constant

Compared to regular state variables, the Gas costs of *constant* variables are much lower.

Path:
./src/contracts/vault/FeeOracle.sol : weightDenominator

Recommendation: Make *weightDenominator* constant.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L08. Missing Validation

Missing *require* check leads to unclear error messages in case of providing incorrect data.

Path:
./src/contracts/vault/FeeOracle.sol : setTargets()

Recommendation: Validate that *weights.length <= 50*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L09. Typos

Any typos encountered in the provided documentation should be addressed.

“weightes” -> “weights”
“disatnce” -> “distance”
“coinPirceUSD -> coinPriceUSD”

“fomula -> formula”
“denomiator -> denominator”
“decominator -> denominator”

Paths:
./src/contracts/vault/FeeOracle.sol
./src/contracts/vault/Vault.sol

Recommendation: Fix typos.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L10. Functions That Can Be Declared External

Public functions that are never called by the contract should be declared *external* to save Gas.

Path:
./src/contracts/vault/FeeOracle.sol : isInTarget()

Recommendation: Use the external attribute for functions never called from the contract.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L12. Unused Variable

Unused variables should be removed from the contracts. Unused variables are allowed in Solidity and do not pose a direct security issue. It is best practice to avoid them as they can cause an increase in computations (and unnecessary Gas consumption) and decrease readability.

Path:
./src/contracts/AddressRegistry.sol : supportedCoinAddresses

Recommendation: Remove unused variables.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L13. Redundant Payable

payable modifier is redundant for *deposit* and *withdraw* functions, as *Vault* contract implements them as not payable.

Path:
./src/contracts/Router.vy : IVault.deposit, IVault.withdraw

Recommendation: Remove redundant payable.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L14. Redundant Iterations

getCoinPositionInCPU function will iterate 50 times, in case *len(cpu)* is less than 50 and coin position in CPU does not exist. Such iteration could be optimized by limiting the range to *len(cpu)*.

Path:
./src/contracts/Router.vy : getCoinPositionInCPU()

Recommendation: Replace *range(50)* with *range(len(cpu))*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L15. Code Duplication

DepositParams and *WithdrawalParams* structs are identical, so could be merged into a single, reusable struct.
The same is applicable for *DepositFeeParams* and *WithdrawalFeeParams*.

During the second review, *DepositParams* and *WithdrawalParams* structs are duplicated.

Paths:
./src/contracts/vault/Vault.sol : DepositParams, WithdrawalParams
./src/contracts/structs/structs.sol : DepositFeeParams, WithdrawalFeeParams

Recommendation: Remove code duplication.

Found in: 85abc9e

Status: Fixed (Revised commit: 3bb9ee7)

L16. Access Control Violation

The function *initialize()* is called when the current contract has no owner, without additional protection.

As a consequence, this function can be called by any actor, resulting in a contract takeover.

Path:
./src/contracts/Router.py : initialize()

Recommendation: It is recommended to use a constructor instead.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L17. Misleading Function Names

The code contains the following functions, whose names are not representative of their behavior: *getWhitelistedStrategies()* should be similar to *isWhitelistedStrategy()*, and *getWhitelistedRebalancer()* similar to *isWhitelistedRebalancer()*.

Path:
./src/contracts/AddressRegistry.sol: getWhitelistedStrategies(), getWhitelistedRebalancer().

Recommendation: It is recommended to use names that reflect the function purpose as closely as possible to increase readability.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L18. Explicit Uint Size

The function *getComponentAmount()* returns a *uint* of non-explicit size.

The mapping *blockCapCounter* uses keys of non-explicit *uint* size.

Paths:
./src/contracts/vault/Vault.sol : blockCapCounter
./src/contracts/strategy/IStrategy.sol : IStrategy.sol: getComponentAmount()

Recommendation: It is recommended to explicitly declare the size of *uint* variables.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L19. Inefficient Gas Model

The function *isInTarget()* iterates over the *storage* variable *targetsLength*.

This approach uses a lot of Gas. Instead, a new *memory* variable should be declared and used into the loop.

Path:
./src/contracts/vault/FeeOracle.sol: isInTarget().

Recommendation: Load the value of *targetLengths* into a new *memory* variable and use it in the loop.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L20. Missing Assert Messages

In Router.vy, *assert* is used to enforce certain conditions. However, no error message is provided.

Path:
./src/contracts/Router.vy

Recommendation: Consider adding error messages in the assert checks of the contract.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L21. Redundant Code

The function *reinitialize* re-sets the variable *self.owner* as *msg.sender*, resulting in no change and thus spending Gas unnecessarily.

Path:
./src/contracts/Router.vy: reinitialize().

Recommendation: It is recommended to delete the statement *self.owner = msg.sender*.

Found in: 85abc9e

Status: Fixed (Revised commit: 7927f31)

L22. Inefficient Gas Model

The function *processMintRequest()* checks validity of fee and *feeDenominator* during it's execution, which increases Gas spendings. This logic could be moved to *setFee* / *setFeeDenominator* functions and validated once during initialization.

Path:
./src/contracts/Router.vy: processMintRequest(), processBurnRequest()

Recommendation: Move check to the setter functions.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

L23. Redundant Code

Router contract does not support native tokens (*address(0)*); however, *Vault* contract still processes *address(0)* as native tokens. *claimDebt* function is called only by router, so zero address for coin is not possible. This code is unreachable and could be removed.

Consider checking all payable functions in Vault contract if they are required.

Path:
./src/contracts/vault/Vault.sol : claimDebt();

Recommendation: Remove unused code. Check other payable functions in the *Vault* contract.

Found in: 7927f31

Status: Fixed (Revised commit: 3bb9ee7)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.